Tampereen yliopisto

Riku Ketonen

NYKYAIKAISTEN SALATTUJEN TIEDONSIIRTO-PROTOKOLLIEN VERTAILU

Tekniikan ja luonnontieteiden tiedekunta
Diplomityö
Elokuu 2019

TIIVISTELMÄ

Riku Ketonen
Nykyaikaisten salattujen tiedonsiirtoprotokollien vertailu
Tampereen yliopisto
Tietoturvan ja johtamisen DI-linja
Diplomityö
Elokuu 2019

Räjähdysmäisesti kasvava Internet-käyttäjien määrä on luonut uusia tarpeita nopeille ja salatuille yhteyksille. Tätä ongelmaa lähti ratkomaan Internet Engineering Task Force, joka neljän vuoden ajan kehitti uutta versiota Transport Layer Security eli TLS-protokollalle. Työ oli avointa ja siihen osallistui moni nimekäs yritys, mutta myös kryptoharrastajat.

Tämän työn tarkoituksena oli tutkia uusinta TLS-protokollan versiota erityisesti nopeutensa puolesta sekä samalla tutustua sen toimintoihin syvemmin. Paljon puhutusta yhteyden luomisen nopeutumisesta haluttiin saada tieteellisiä tuloksia ja vertailla niitä aiempiin versioihin. Protokollan teoriaa ja kehitystyötä tutkittiin paljon, joka antoi lähtökohdat protokollan suorituskykyodotuksille. Salatun yhteyden muodostamiseen kuuluvan kättelyn odotettiin olevan noin 50% nopeampaa aiempiin versioihin verrattuna, ja salaukseen käytettävien algoritmien, funktioiden ja metodien listan odotettiin olevan lyhyempi kuin aiemmin. Protokollien vertailu suoritettiin laboratorio-oloissa, jossa ulkopuoliset haitat saatiin poistettua ja muuttujia kyettiin määrittelemään tarpeiden mukaan.

Työn tulokset olivat odotuksien mukaiset. TLS-protokollan versio 1.3 onnistui pudottamaan salatun yhteyden luomiseen vaaditun kättelyn pituuden puoleen. Tuettujen algoritmien, metodien ja funktioiden lista on saatu tiivistettyä luotettaviin vaihtoehtoihin. Osa aiemmin käytetyistä vaihtoehdoista oli kyseenalaistettu kryptoyhteisön toimesta. Tutkimus osoitti protokollan uuden version olevan valmis laajempaan yleiseen käyttöön.

Avainsanat: Internet, salaus, TLS, transport layer security

ABSTRACT

Riku Ketonen
Comparison of Modern Encryption Communication Protocols
Tampere Universities
Degree Programme in Management and Information Technology, MSc (Tech)
Master's thesis
August 2019

The growing number of Internet users has created new needs for fast and encrypted connections. To solve this problem, Internet Engineering Task Force started the development project for a new version of TLS, Transport Layer Security, which lasted for four years. The project was open for anyone, and many big Internet companies and crypto enthusiasts participated in it.

The purpose of this work was to examine the latest version of TLS, especially for its speed, and at the same time to familiarize in its features. The much-talked-about increase in its connection speed was the main focus, along with comparing it to the older versions. The theory and research put into this protocol was studied closely, which provided the starting point for expectations: the handshake was expected to be around 50% faster compared to previous versions, and the list of encryption algorithms, functions and methods was expected to be much shorter than before. The comparisons of the protocols were carried out in laboratory conditions where external noise was eliminated, and variables could be defined accordingly.

The results were much as expected. The version 1.3 of the TLS protocol succeeded in dropping the length of the handshake required for encryption, and the list of supported algorithms, methods and functions had been compiled into reliable alternatives, as some of the previously used options were challenged by the crypto community. The study proved that the new version of the protocol was ready for wider public use.

Keywords: Internet, encryption, TLS, transport layer security

ALKUSANAT

Tällä sivulla on otettu vapauksia tieteellisestä kirjoitusasusta.

Tämän työn alkuperäinen tarkoitus oli vain mahdollistaa valmistumiseni, mutta loppua kohden innostuin tutkimaan asioita enemmän ja syvemmältä kuin alun perin oli tarkoitus. Tutkimuksen parissa kului useampi etäpäivä töistä sekä vapaata vasten olleet yöt pitkälle aamun pikkutunneille, mukaan lukien juuri tämä hetki.

Kiitos Tampereen Yliopisto, omaa sukuaan Tampereen Teknillinen Yliopisto.

Kiitos professoreille, jotka kultivoivat uteliaisuuttani tietoturvaa ja sen johtamista kohtaan.

Erityiskiitokset työni ohjaajalle Juha Vihervaaralle, joka jaksoi ohjata ja korjata tötöilyjäni työn parissa, sekä työn viimeiselle tarkastajalle ja yliopistokeskuksen johtajalle, Tarmo Lippingille.

SISÄLLYSLUETTELO

TERMIT JA MÄÄRITELMÄT

AES	*Advanced Encryption Standard*, alkuperäiseltä nimeltään Rijndael, on lohkosalausmenetelmä, joka luotiin NIST:n järjestämän kilpailun voittajaehdokkaan pohjalta.
Bitti	*Bitti* on tietotekniikassa pienin mahdollinen yksikkö, jolla kuvataan informaatiota. Bitti voi olla arvoltaan vain 0 tai 1.
CBC	*Cipher Block Chaining*, lohkosalausketjutus, on menetelmä, jolla salattava tieto salataan saman kokoisissa paloissa.
ChaCha20	Daniel J. Bernsteinin vuonna 2008 julkaisema salaussarja *ChaCha*, joka perustuu aiempaan salaussarjaan *Salsa*.
Cipher	Salaus, koodaus. Tarkoituksena estää datan lukeminen vapaasti.
Cipher Spec	Yhdistelmä salausalgoritmista ja viestinvahvistuskoodista.
Cipher Suite	Salaussarja. Lista avaimenvaihtoon ja varmistukseen käytettävästä algoritmista, salausalgoritmista sekä viestinvarmistuskoodista.
Client	Käyttäjä, tai sovellus, joka pyytää data palvelimelta.
CRC	*Cyclic Redundancy Check*, eli syklinen redundanssintarkastus, on metodi, jolla tarkastetaan bittiryhmän muuttumattomuus. Näin estetään tiedon vikaantuminen virheelliseksi.
Curve25519	Elliptinen käyrä, joka tarjoaa 128 bitin salauksen ja on suunniteltu käytettäväksi Diffie-Hellmanin avaimenluontimetodissa. Se on yksi nopeimmista patentoimattomista ECC-käyristä.
Decryption	Dekryptaus, eli salauksen purku, tarkoittaa salatekstin muuttamista takaisin luettavaan tai käsiteltävään muotoon.
DH	*Diffie-Hellman*, metodi salausavainten vaihtoon.
Digitaalinen allekirjoitus	Digitaalinen allekirjoitus on matemaattinen tapa luoda vahva tunniste lähetetylle viestille. Viestin tunniste voidaan luoda vain lähettäjän yksityisellä avaimella, ja se voidaan avata lähettäjän julkisella avaimella. Tällä tavoin voidaan varmistua siitä, että viesti on pysynyt muuttumattomana ja on alkuperäiseltä lähettäjältä.
DSA	*Digital Signature Algorithm*. Katso edellinen.
ECC	*Elliptical-curve Cryptography*, eli elliptisen käyrän kryptografia. Elliptisiä käyriä voidaan käyttää avainten luomiseen, digitaaliseen allekirjoitukseen, näennäissatunnaiseen lukugeneraattoriin sekä muihin matemaattisiin tarkoituksiin. Käytetään yleensä symmetrisen avaimen luontiin.
ECDHE	*Elliptical-curve Diffie-Hellman*, eli avaimenvaihtoprotokolla, joka mahdollistaa kahden osapuolen avaintenvaihdon tukeutuen elliptisen käyrän avulla luotuihin yksityisiin ja julkisiin avaimiin.

ECDSA	*Elliptic Curve Digital Signature Algorithm,* elliptiseen käyrään perustuva digitaalisen allekirjoituksen algoritmi.
Encryption	Enkryptaaminen, eli salaaminen. Dekryptauksen vastainen toimenpide, tarkoituksena muuttaa luettava tai käsiteltävä tieto salatekstiksi, jotta ulkopuoliset eivät voi sitä lukea.
FTP	*File Transfer Protocol*, tiedoston siirtoprotokolla, on luotu siirtämään tiedostoja verkon yli.
GCM	*Galois/Counter Mode* on symmetrisen avaimen lohkosalaukseen liittyvä operaatio, jota käytetään tehokkuutensa ja suorituskykynsä vuoksi. Operaatio on suunniteltu todentamaan datan eheyden ja luottamuksellisuuden. GCM on määritelty 128-bittisille lohkoille.
Heksadesimaali	Heksadesimaalijärjestelmä on 16-kantainen lukujärjestelmä. Tämä tarkoittaa sitä, että yhdellä merkillä voidaan esittää luvut 0-15 välillä käyttäen 0-9 sekä A-F merkintöjä. Heksadesimaaleja käytetään erityisesti tietotekniikassa, sillä yksi heksadesimaali vastaa neljää peräkkäistä bittiä.
HTTP	*Hypertext Transfer Protocol* on protokolla, jota käytetään tiedonsiirtoon WWW-palvelimien ja sovelluksen välillä.
HTTPS	*Hypertext Transfer Protocol Secure* on salattu versio HTTP-protokollasta. Tiedonsiirto salataan toisten protokollien, kuten SSL:n tai TLS:n, avulla.
IANA	*Internet Assigned Numbers Authority* on amerikkalainen voittoa tavoittelematon viranomainen, joka valvoo maailmanlaajuista IP-osoitteiden jakamista, autonomisen järjestelmänumeron allokointia, juurivyöhykkeiden hallintaa verkkotunnusjärjestelmässä (DNS), mediatyypejä ja muita Internet-protokolliin liittyviä symboleja ja numeroita
IETF	*Internet Engineering Task Force* on organisaatio, joka vastaa Internet-protokollien standardisoinnista. Suurin osa Internetissä käytetyistä protokollista on IETF:n aikaansaannoksia.
Kryptografia	Muinaisesta kreikasta otettu lainasana. Suora käännös on “Salakirjoitus”, mutta nykypuheessa sillä usein viitataan salaukseen tai tekstin muokkaamiseen muotoon, jota ei voida lukea ilman apuvälineitä.
MAC ja HMAC	*Message Authentication Code* ja *Hash-based Message Authentication Code* ovat viestin varmistukseen käytettäviä koodeja, jotka varmistavat viestin oikeellisuuden ja muuttumattomuuden tiivistefunktioiden avulla.
MD5	*MD5 (Message-digest algorithm 5)* on tiivistefunktio, joka tuottaa 128-bittisen tiivisteen annetusta syötteestä. Alun perin suunniteltu kryptografiseksi tiivistefunktioksi, mutta myöhemmin todettu turvattomaksi.
OSCP	*Online Certificate Status Protocol* on protokolla, jota käytetään tarkastamaan digitaalisten X.509 -sertifikaattien voimassaolo. Sertifikaatti voidaan poistaa käytöstä, jolloin sen tilaksi muuttuu "Revoked" ja OSCP-tarkastus ilmoittaa tästä.
OSI-malli	*Open Systems Interconnection Reference Model* on malli, joka kuvaa tiedonsiirtoprotokollien järjestystä seitsemällä eri

	kerroksella. Malli kuvaa tiedon muuttumista signaalista sovellusdataksi asti.
Poly1305	*Poly1305* on Daniel J. Bernsteinin luoma algoritmi, jolla voidaan varmistua viestin oikeellisuudesta ja muuttumattomuudesta kuten MAC:llä ja HMAC:llä.
PSK	*Pre-Shared Key* on kahden osapuolen välillä jaettu salaisuus, josta voidaan muodostaa symmetrinen avain.
PSS	*Probabilistic Signature Scheme* on todennäköisyyksiin perustuva allekirjoitusmenetelmä, jonka tarkoitus on korvata RSA-PKCS#1 v1.5 viestien allekirjoituksessa.
RC4	*Rivest Cipher 4 (myös ARC4, ARCFOUR)* on yksinkertainen ja nopea virtaussalausalgoritmi, josta on löydetty useita haavoittuvuuksia. Algoritmi suunniteltiin vuonna 1987, ja se vuosi julkisuuteen ensimmäisen kerran 1994.
RFC	*Request For Comments* on dokumentaatiomuoto, jota suosivat erityisesti Internet Engineering Task Force, Internet Research Task Force sekä Internet Architecture Board. RFC-dokumentaatio on usein informatiivista tai kokeellista.
RSA	*Rivest-Shamir-Adleman* on yksi ensimmäisistä julkiseen avaimeen pohjautuvista kryptojärjestelmistä. Tässä järjestelmässä salausavain on julkinen ja purkamiseen tarvittu avain on salainen.
RTT	*Round-trip-time* on aika, joka datapaketilla kestää kulkea yhteyden päästä päähän ja takaisin alkupisteeseensä.
Salaus, Asymmetrinen	*Asymmetric encryption*. Viestin salaamiseen ja purkamiseen käytetään eri avaimia.
Salaus, Symmetrinen	*Symmetric encryption*. Viestin salaamisessa ja purkamisessa voidaan käyttää samaa avainta.
Salsa20	*Salsa20* on virtaussalain, joka suunniteltiin vuonna 2005. Tästä on myöhemmin muokattu *ChaCha20*.
SCTP	*Stream Control Transmission Protocol* toimii OSI-mallin siirtokerroksella. Protokollan tarkoituksena on tarjota ominaisuuksia sekä TCP:lle että UDP:lle ja estää verkkoa tukkeutumasta.
Sertifikaatti	Sertifikaatti on todistus, jolla voidaan varmistaa osapuolen identiteetti.
Server, Palvelin	Laite, joka palvelee useaa käyttäjää yhtä aikaa erinäisin palveluin, esimerkiksi verkkosivuja tarjoamalla.
Session	Session tai istunto on käytäntö, jolla luodaan pysyvä yhteys osapuolten välille.
SHA-1	*Secure Hash Algorithm 1* on kryptografinen tiivistefunktio, joka ottaa syötteen ja tuottaa siitä 160-bittisen tiivisteen, joka tunnetaan myös hajautusarvona.
SHA-256	*Secure Hash Algorithm 256* toimii kuten *SHA-1*, mutta luo pidemmän tiivisteen kuin edeltäjänsä. Kts. edellinen.
SSL	*Secure Socket Layer*, protokolla salatun yhteyden luomiseen ja käyttämiseen.

Tavu	Mittayksikkö, joka sisältää kahdeksan (8) bittiä.
TCP	*Transmission Control Protocol* on protokolla, jonka avulla luodaan yhteyksiä tietokoneiden välille. Protokolla pitää huolta pakettien järjestyksestä ja tarvittaessa lähettää hävinneen paketin uudelleen. Jokainen paketti kuitataan vastaanottajan toimesta saapuneeksi, muussa tapauksessa paketti lähetetään uudelleen.
TLS	*Transport Layer Security*. Uudempi versio SSL:stä, mutta molempien periaate on sama.
UDP	*User Datagram Protocol* on epäluotettava protokolla, joka ei vaadi laitteiden välille yhteyttä.
x25519	Curve25519:ssa käytettävä Diffie-Hellman funktio, joka on Daniel J. Bernsteinin käsialaa.

1 JOHDANTO

Internetin tietoturva on ollut tapetilla jo pitkän aikaa. Erilaisia hyökkäyksiä on toteutettu suojaamattomiin sekä suojattuihin yhteyksiin. Yritysten välisten sopimusten luomia skandaaleja on tullut julkisuuteen. Monenlaisia tietoturvarikkeitä päätyy uutisiin joka viikko. Varastettuja identiteettejä ja avoimeksi jääneitä palveluita, tietokantoja ja palvelinympäristöjä löydetään lähes päivittäin. Näiden ongelmakohtien lomassa on kuitenkin ollut paljon kehitystä.

Yli neljä vuotta kehityksessä ollut Transport Layer Security-protokollan versio 1.3 sai Internet Engineering Task Force:n hyväksynnän vihdoin kesällä 2018 [1]. Uuden protokollan tarkoituksena oli nopeuttaa verkon toimintaa ja parantaa turvallisuutta poistamalla vanhoja ominaisuuksia. Muutos oli kiinnostava, sillä ensimmäistä kertaa protokollaa pyrittiin parantamaan tietoturvayhteisön avulla eikä vain yksittäisten ryhmittymien toimesta. Vuonna 2016 järjestetyssä IETF 95 -tapahtumassa CloudFlare ja Mozilla implementoivat sen hetkisen version TLS 1.3:sta järjestelmiinsä ja haastoivat tietoturvasta innostuneet ottamaan osaa protokollan implementaatioon ja murtamisyrityksiin.

Tämän työn tarkoituksena on verrata TLS 1.3 -protokollaa vanhempiin versioihin keskittyen erityisesti verkkosivujen lataamiseen kuluvaan aikaan. Versioiden näkyvimmät erot liittyvät yhteyksien muodostamiseen, johon kuluvaan aikaan voidaan vaikuttaa riippumatta verkkosivusta, palvelimesta tai sovelluksesta. Samaiseen yhteydenmuodostusaikaan vaikuttavat myös HTTP-protokollan eri versiot, joiden vaikutus otettiin myös huomioon.

Työssä tuodaan esille eri protokollien tuomia säästöjä verkon resurssien käytössä, mutta myös peruskäyttäjän kokemia parannuksia. Nopeammat latausajat tuntuvat heti sivuston avaamisessa.

Materiaalia ja lähteitä on tarjolla runsaasti. Verkon tietoturvaa on kehitetty pitkään ja hartaasti aina vuodesta 1994 lähtien, kun Netscape otti käyttöön ensimmäisen versionsa SSL-protokollasta. Tieteellisiä julkaisuja ja tutkimustyötä on tehty yhtä pitkään eikä vauhti ole hidastumassa, sillä jokaista uutta protokollaa kohden tuotetaan 100-200 sivua yleistä dokumentaatiota [2, 3], joihin viitataan tuhansissa julkaisuissa. TLS 1.2 -

protokollan dokumentaatiota on siteerattu yli 4000 kertaa [4]. Julkaisujen aiheet vaihtelevat eri käyttötavoista aina haavoittuvuuksiin asti [5].

Tutkimuksen tavoitteena oli tarkastella nykyaikaisia tietoturvaprotokollia käytännössä sekä vertailla erilaisia käyttötilanteita, joihin normaali käyttäjä saattaisi ajautua jokapäiväisen verkon käytön yhteydessä. Nämä käyttötilanteet rajattiin tarkastelemalla peruskäyttäjän toimia verkon parissa muutaman minuutin ajalta. Yleisimmäksi käyttötavaksi todettiin verkkosivustojen lataaminen HTTP:n yli palvelimilta, jotka saattoivat sijaita toisella puolen maailmaa.

Suunnitelmana työn toteutukselle oli luoda suljettu verkko ja dedikoida erilliset laitteet käyttötilanteiden mittausta varten. Suljettu verkko luotiin Porin yliopistokeskuksen tiloihin, josta löytyi jo valmis sisäverkko sekä vapaita laitteita palvelinkäyttöä varten. Laitteet tyhjennettiin kaikesta aiemmasta datasta, jonka jälkeen niille asennettiin tarvittavat sovellukset etäkäyttöä ja virtualisointia varten. Virtualisointisovelluksen päälle asennettiin tarvitut käyttöjärjestelmät, joiden päälle asennettiin mittaukseen käytettävät sovellukset. Mittaukset suunniteltiin aiemmin toteutetun käyttäjätarkastelun avulla, jonka perusteella päätettiin työssä käytettävät protokollat sekä simuloitavat etäisyydet palvelimille. Mittaukset aloitettiin loppukesästä vuonna 2018, kun TLS 1.3 vihdoin julkaistiin ja tarvitut sovellukset saatiin päivitettyä yhteensopiviksi.

Työssä kuvataan ensin tarvittu teoria lähtien aina OSI-mallin tasoista, joista siirrytään yhteydenluontiprotokollaan, yhteyden yli siirrettävään dataan ja lopulta datan salaukseen. Teoriaa seuraa työn käytännön osuus, joka pitää sisällään laboratorioympäristössä toteutetun mittatyön esittelyn, mittaustulokset sekä tulosten vertailun. Lopulta pohditaan työn tavoitteiden toteutumista, mitä hyötyä työstä muodostui sekä miten työtä olisi voitu parantaa.

2 TIETOLIIKENNEPROTOKOLLAT

Tietoliikenneprotokollia on useita erilaisia, ja jokaisella on oma tarkoituksensa dataa siirtäessä. Open Systems Interconnection -malli (OSI-malli) on käsitteellinen malli, joka luonnehtii ja standardoi tietoliikenne- tai tietojenkäsittelyjärjestelmän viestintätoimintoja ottamatta huomioon sen sisäistä rakennetta ja teknologiaa. Sen tavoitteena on erilaisten viestintäjärjestelmien yhteensopivuus vakioprotokollien kanssa. Malli jakaa viestintäjärjestelmän abstraktiokerroksiksi. Mallin alkuperäinen versio määrittelee seitsemän kerrosta, jotka näkyvät Kuvassa 1.

#	OSI-kerros	TCP/IP	Yksikkö	Toiminto	Esimerkki
7	Sovellus	Sovellus	Data	Palvelut, joita loppukäyttäjä käyttää	SMTP
6	Esitystapa			Muuttaa datan käyttäjälle luettavaksi. Salaa ja purkaa	JPG, GIF, HTTPS, SSL, TLS
5	Istunto			Luo/lopettaa pysyvän yhteyden kahden päätepisteen välillä	NetBIOS, PPTP
4	Kuljetus	Siirto	Segmentti	Vastuussa siirtoprotokollasta sekä virheiden käsittelystä	TCP, UDP
3	Verkko	Internet	Paketti / Datagram	Lukee paketin sisältämän IP-osoitteen ja porttitiedot, reititys	Reititin, kytkin
2	Siirto	Verkkoliittymä	Bitti / Frame	Luotettavan yhteyden luonti kahden päätepisteen välille	Kytkin
1	Fyysinen		Bitti	Lähettää datan siirtoyhteydelle	Hub, verkkokortti, kaapeli

Kuva 1. *OSI-mallin kerrokset [6].*

Tietty kerros palvelee sen yläpuolella olevaa kerrosta, ja sitä palvelee sen alla oleva kerros. Kaksi samassa kerroksessa olevaa esiintymää visualisoidaan niin, että ne on yhdistetty vaakasuoraan yhteyteen kyseisessä kerroksessa [7].

Alimmainen kerros OSI-mallista on fyysinen kerros, joka käsittää kaikki mekaanisiin ja sähköisiin ominaisuuksiin liittyvät asiat. Fyysisellä tasolla tieto siirretään sarjamuotoisesti tai rinnakkaismuotoisesti. Sarjamuotoisessa vaihtoehdossa bitit on siirrettävä peräkkäin yksi kerrallaan. Yksisuuntaista siirtoa varten siirtojohtimia tarvitaan vain kaksi, tai vaihtoehtoisesti kolme, mikäli halutaan kaksisuuntaista siirtoa. Bitit kuljetetaan peräkkäin, joten ne tulee myös merkitä jotenkin. Tällä tavoin bitit ja tavut voidaan erottaa toisistaan.

Rinnakkaismuotoisessa tiedonsiirrossa kaikki yhden merkin bitit siirretään samaan aikaan, jokainen omaa johdintaan pitkin. Sarjamuotoiseen siirtoon verrattuna rinnakkaismuotoisen siirron nopeus on moninkertainen. Liipaisujohtimen signaalia

voidaan käyttää ilmoittamaan uudesta merkistä. Rinnakkaisten johtimien määrä rajoittaa järjestelmän kannattavuutta pitkillä matkoilla [8].

Toinen kerros, eli siirtokerros, hoitaa yhteyden luomisen, virheiden korjaamisen ja yhteyden purkamisen. Yhteyden luominen ja purkaminen toteutuvat fyysisen kerroksen mukaan. Kupariyhteyksien ja langattomien yhteyksien luominen ja purkaminen tapahtuvat eri tavalla. Siirtokerros säätelee lähetysnopeutta. Tämä varmistaa, ettei vastaanottaja tukehdu saapuvan datan määrään. Kerros myös varmistaa datan virheettömyyden. Varmistaminen tapahtuu erilaisin virhekoodein ja tarpeen tullen data lähetetään uudelleen [8].

Verkkokerroksen ominaisuudet luovat verkolle toteutuksesta riippumattoman tiedonsiirron mahdollisuuden. Kerros piilottaa fyysiseen toteutukseen liittyvät aspektit, koska fyysisesti eri tavalla rakennetut verkot voivat olla toiminnaltaan kuitenkin samanlaisia. Kerros valitsee, mitä reittiä pitkin tieto siirretään, mikäli vaihtoehtoja on useampia [8].

Kuljetuskerros takaa luotettavan yhteyden päästä päähän. Mikäli lähetetty data katoaa, kuljetuskerroksen tehtävänä on huolehtia datan perillemenosta ilman, että tietoliikenne katkeaa. Kuljetuskerros on vastuussa virtauksen hallinnasta (flow control), datan segmentoinnista ja palauttamisesta [8]. TCP sekä UDP ovat kuljetuskerroksen protokollia [6].

Istuntokerros tarjoaa tarvittavat mekanismit pysyvän istunnon luomiseen, sulkemiseen ja ylläpitämiseen. Istuntokerros myös synkronoi datavirtoja, kuten videoiden äänen ja kuvan [9].

Esitystapakerros, toisinaan myös syntaksikerros, on vastuussa informaation muuttamisesta sovelluksille käytettävään muotoon. Esitystapakerros on myös vastuussa datan salaamisesta ja purkamisesta [10]. Esitystapakerros sisältää mm. SSL:n ja TLS:n. [6]

Sovelluskerros on OSI-mallin ylin kerros. Se tarjoaa palvelut, joiden avulla sovellukset yhdistetään alemmille kerroksille ja varmistaa tehokkaan viestinnän muiden sovellusten kanssa verkon yli [11]. Sovelluskerros sisältää esimerkiksi HTTP-protokollan [6].

2.1 TCP

Transmission Control Protocol on yksi TCP/IP-protokollaperheen tärkeimmistä protokollista. Näitä protokollia käytetään eri osapuolten yhdistämiseen Internetiin ja näiden osapuolten keskinäiseen kommunikointiin tietokoneverkkojen yli.

Protokolla on yhteisesti sovittu muoto jonkin asian toteuttamiseen. Tietokoneiden osalta sitä käytetään yleisimmin viittaamaan tiettyihin sääntöihin (eli standardiin), joiden avulla tietokoneet voivat muodostaa yhteyden toisiinsa ja lähettää tietoja toisillensa. Tätä kutsutaan myös viestintäprotokollaksi.

TCP on yhteyspohjainen viestintäprotokolla, joka tarkoittaa sitä, että se muodostaa ja ylläpitää virtuaalista yhteyttä isäntien välillä, kunnes siihen liittyvän sovellusohjelman viestit on vaihdettu. Se jakaa kaikki lähetettävät viestit paketeiksi, numeroi ne ja välittää ne erikseen IP-ohjelman tasolle. Vaikka kullakin paketilla on sama kohde-IP-osoite, ne voivat saada eri reitityksen verkon yli.

TCP käyttää virheenkorjaus- ja virranhallintatekniikoita varmistaakseen, että paketit saapuvat tarkoitettuihin kohteisiinsa korruptoimattomina ja oikeassa järjestyksessä. Tämä tekee pisteestä pisteeseen yhteyden lähes virheettömäksi. Paketit ovat TCP/IP-verkoissa tapahtuvan tiedonsiirron perusyksikköjä.

TCP toimii kuljetuskerroksessa, eli siis OSI:n neljännessä kerroksessa. Tämä kerros on vastuussa luotettavan viestinnän ylläpidosta kahden eri pisteen välillä verkon yli. IP on sen sijaan verkkokerroksen protokolla, joka on kerros juuri kuljetuskerroksen alapuolella. Kuljetuskerroksella toimivat myös UDP (User Datagram Protocol), RTP (Real-Time Transport Protocol) sekä SCTP (Stream Control Transmission Protocol).

Useimmat sovellusprotokollat, jotka edellyttävät luotettavaa tietojen siirtoa, käyttävät TCP:tä. Näitä ovat esimerkiksi HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), sekä IMAP (Internet Message Access Protocol) [12, 13].

2.1.1 Yhteyden muodostus

Yhteyden muodostamiseksi TCP käyttää kolmivaiheista kättelyä. Ennen kuin sovellus yrittää muodostaa yhteyden palvelimeen, palvelimen on yhdistettävä itsensä paikalliseen porttinumeroon ja kuunneltava sitä. Tätä kutsutaan passiiviseksi avaamiseksi. Kun passiivinen avaaminen on suoritettu, sovellus voi aloittaa aktiivisen avaamisen. Yhteyden muodostamiseksi tapahtuu kolmivaiheinen kättely:

- SYN: Sovellus suorittaa aktiivisen avaamisen lähettämällä SYN-viestin palvelimelle. Sovellus asettaa segmentin sarjanumeron satunnaiseen arvoon “A”.
- SYN-ACK: Vastauksena palvelin vastaa SYN-ACK-viestillä. Kuittausarvona sovellukselle lähetetään “A+1” suuruinen arvo. Lisäksi palvelin asettaa kehykseen oman sekvenssinumeron “B”.
- ACK: Lopuksi sovellus lähettää ACK-viestin takaisin palvelimelle. Järjestysnumero “A” asetetaan vastaanotettuun kuittausarvoon “A+1”, ja vastaanotettu sekvenssinumero “B” kasvatetaan arvoon “B+1”.

Tässä vaiheessa sekä sovellus että palvelin ovat saaneet kuittauksen yhteydestä. Vaiheet yksi ja kaksi muodostavat yhteyden parametrit (sekvenssinumerot) yhdelle suunnalle. Vaiheet kaksi ja kolme muodostavat vastakkaisen yhteyden parametrit. Tällä tavoin muodostetaan full-duplex eli täysin kaksisuuntainen TCP-yhteys. Tämän jälkeen datan siirto voi alkaa [14].

2.1.2 Yhteyden purku

Yhteyttä suljettaessa käytetään nelivaiheista kättelyä, jolloin kummankin puolen yhteys päättyy itsenäisesti. Kun päätepiste haluaa lopettaa oman puolensa yhteydestä, se lähettää FIN-paketin, jonka toinen pää kuittaa ACK-paketilla. Tästä syystä tyypillinen purku vaatii FIN-ACK-parin kummaltakin osapuolelta. Kun FIN-paketin lähettänyt osapuoli on vastaanottanut ACK-pakettiin, se odottaa aikakatkaisua ennen yhteyden lopullista sulkemista. Tämä pitää paikallisen portin varattuna eikä anna luoda uusia yhteyksiä sille. Tällä tavoin mahdolliset sekaannukset myöhästyneiden pakettien johdosta estetään.

Yhteys voi myös olla “puoliksi avoin”, jolloin vain toinen puoli yhteyttä on päätetty. Lopettanut osapuoli ei voi enää lähettää data yhteyteen, mutta toinen puoli voi. Lopettaneen osapuolen tulisi yhä lukea data yhteydeltä, kunnes vastapuolikin lopettaa.

Yhteyden voi myös lopettaa kolmivaiheisella kättelyllä. Kun osapuoli A lähettää FIN-paketin, vastapuoli B lähettää vastaukseksi FIN & ACK -paketin yhdistäen kaksi vaihetta yhdeksi. Lopuksi A vastaa jälleen ACK-paketillaan [15].

TCP hallitsee yhteyksiä käyttöjärjestelmäpuolen taulukkorakenteella. Koska TCP-paketit eivät sisällä istunnon tunnistetta, molemmat päätepisteet tunnistavat istunnon sovelluksen osoitteen ja portin avulla. Kun paketti on vastaanotettu, TCP-toteutuksen on käytävä taulukko läpi kohdeprosessin löytämiseksi. Jokainen taulukon merkintä tunnetaan lähetysohjauslohkona, tai TCB:nä (Transmission Control Block). Se sisältää

tiedot päätepisteistä (IP-osoite ja portti), yhteyden tilasta, vaihdettavien pakettien tiedoista ja puskureista datan lähettämiseen ja vastaanottamiseen [16].

2.2 HTTP

HTTP (Hypertext Transfer Protocol) on sovellustason protokolla hajautetuille hypermedian järjestelmille. HTTP on ollut käytössä vuodesta 1990 lähtien World-Wide Web Global Information -aloitteesta. HTTP:n ensimmäinen versio HTTP/0.9 oli yksinkertainen protokolla raakaa tiedonsiirtoa varten Internetin yli. RFC-1945:n määrittelemä HTTP/1.0 paransi protokollaa sallimalla viestien olevan MIME-määrittelyn kaltaisia sisältäen metainformaatiota siirretyistä tiedoista ja muuttujista. Sen perustana oli Request/Response -semantiikka. HTTP/1.0 ei kuitenkaan ottanut riittävästi huomioon välityspalvelinhierarkiaa, välimuistia, tarvetta pysyville yhteyksille tai virtuaalisille palvelimille. Lisäksi epätäydelliset itseään HTTP/1.0:ksi kutsuvat sovellutukset aiheuttivat tarpeen muuttaa protokollaversiota uudempaan, jotta kaksi keskenään kommunikoivaa sovellusta voisivat määrittää toistensa todelliset ominaisuudet.

Käytännön tietojärjestelmät vaativat enemmän toimintoja kuin yksinkertaiset palautukset sekä haku, front-end päivitykset ja annotaatiot. HTTP sallii avoimet metodit ja otsakkeet, jotka osoittavat pyynnön tarkoituksen. Se perustuu Uniform Resource Identifierin (URI) referenssiin resurssin sijainnista (URL) tai nimestä (URN), johon metodia sovelletaan. Viestit välitetään sähköpostin käyttämässä Multipurpose Internet Mail Extensions (MIME) -muodossa. HTTP:tä käytetään myös yleisenä protokollana tiedonsiirrossa käyttäjäagenttien, välityspalvelimien ja muiden Internet-järjestelmien välillä, mukaan lukien ne, jotka tukevat SMTP:tä, NNTP:tä, FTP:tä, Gopher:ia, ja WAIS -protokollaa. Tällä tavoin HTTP mahdollistaa pääsyn hypermediaresursseihin monin eri sovelluksin [17].

2.2.1 HTTP/1.1

HTTP/1.0 kykeni määrittelemään vain 16 tilakoodia, joille jokaiselle oli varattu numero. 16 tilakoodin suurin rajoitus oli huono raportointitarkkuus, ja tämä sysäsi HTTP/1.1:n kehityksen polulle. HTTP/1.1 julkaistiin 24 tilakoodilla, jotka ratkaisivat aiemmat HTTP/1.0:n sisältämät rajoitukset. Virheilmoitukset tehtiin nopeammin ja virheiden havaitseminen tapahtumahetkellä oli helpompaa.

HTTP/1.1 sisälsi myös varoitusotsakkeen, joka kykeni siirtämään useampia toisarvoisia varoitusilmoituksia. Toisarvoisten varoitusilmoitusten päällimmäisenä tarkoituksena oli informoida käyttäjää mahdollisista ongelmista, vaikka yhteyden luominen olisikin

onnistunut. Ongelmana saattoi olla esimerkiksi puuttuvat käyttäjätunnukset tai puuttuva verkkosivu. HTTP/1.1:n varoitukset jaettiin kahteen luokkaan. Ensimmäinen luokka käytti kolminumeroisen sarjan ensimmäistä numeroa ja toinen luokka taas sarjan viimeistä numeroa. Nykypäivänä puhutaan HTTP-tilakoodeista.

HTTP/1.0 mahdollisti hyvin yksinkertaisen autentikaation. Kaikki data käyttäjänimistä salasanoihin oli selvätekstiä eikä mitään ole salattu. Tämä mahdollisti vakoilun verkkoliikennettä seuraamalla ja analysoimalla. HTTP/1.0 ei myöskään sisältänyt riippuvaisuuksia kuten aikamääreet, joten kaikki kerätyt datapaketit olivat käytettävissä uudelleen toistohyökkäyksissä. HTTP/1.1:n tulo korjasi ongelman ja tarjosi Digest Access -todennukset, joka mahdollisti kertakäyttöarvojen käytön autentikoinnin yhteydessä. Käyttäjätunnus, salasana sekä kertakäyttöarvo salattiin ja lähetettiin verkon yli, mikä oli merkittävästi turvallisempaa kuin selvätekstin lähetys.

HTTP/1.0 vaati uuden TCP-yhteyden jokaista pyyntöä kohden. Tämä muodosti haasteen, sillä jokainen yhteyden luonti vei merkittävästi aikaa ja resursseja hidastaen koko yhteyttä sovelluksen ja palvelimen välillä. Ongelman ratkaisemista varten HTTP/1.1:een kehitettiin pysyvät yhteydet (Persistent Connections) sekä kanavoidut pyynnöt (Pipeline Requests), jotka vähensivät muodostettavien TCP-yhteyksien määrää [18].

HTTP/1.1 toi siis mukanaan monia parannuksia, kuten nopeammat latausajat, vähemmän toistuvaa dataliikennettä ja aiempien yhteyksien uusiokäyttö. Kaikki muutokset löytyvät dokumentista RFC-2616 [19, 20], mutta tärkeimmät ovat lueteltu alla:

- HTTP/1.1 ei virallisesti vaadi HOST-otsaketta, mutta sen lisääminen ei ole haitaksikaan. Monet sovellukset (proxyt) odottavat näkevänsä HOST-otsakkeen protokollasta huolimatta.
- HTTP/1.1 mahdollistaa useamman jatkuvan yhteyden, joka tarkoittaa sitä, että samalla HTTP-yhteydellä voi olla useampi Request/Response -pari.
- HTTP-sovellus voi käyttää OPTIONS-metodia päätelläkseen, mitä ominaisuuksia HTTP-palvelin tukee.
- HTTP/1.1 laajentaa Caching-tukea lisäämällä Entity-tagin. Jos kaksi resurssia ovat samoja, niillä on sama Entity-tag.
- Koodilla “100 – Continue” estetään sovellusta lähettämästä suurta Request-pyyntöä, josta ei olla varmoja, että palvelin pystyisi sitä käsittelemään, tai sallisi sen käsittelyn. Tässä tapauksessa asiakasohjelma lähettää vain otsakkeet. Palvelin vastaa “100 Continue”, mikäli on valmis käsittelemään pyynnön [19].

Parannuksista huolimatta protokolla oli yhä liian hidas, sillä yhä useammat ihmiset käyttivät mobiililaitteita päästäkseen verkkoon eikä verkon käyttö ollut nautinnollista. Yhden verkkosivun lataamiseen saattoi kulua minuutteja 2-4G yhteyksillä ja sivustojen ulkoisten resurssien määrä kasvatti kuorman suuruutta. Vaikka erilaisia selain- ja palvelinpaikkauksia tehtiin tarpeen tullen, oli jokaisella paikkauksella omat miinuspuolensa. Maailma tarvitsi uutta protokollaa kasvaviin tarpeisiin. Tämä johti HTTP/2.0:n kehitykseen [21].

2.2.2 HTTP/2.0

HTTP/2 on HTTP/1.1-protokollan merkittävä päivitys. Se on peräisin aiemmasta Googlen kehittämästä SPDY-protokollasta, jonka pohjalta IETF:n (Internet Engineering Task Force) työryhmä HTTPbis (Hypertext Transfer Protocol Second) on sen luonut [22].

Työryhmä esitteli HTTP/2:n IESG:lle (Internet Engineering Steering Group) ehdotettuna standardina joulukuussa 2014 [23], ja IESG hyväksyi sen julkaistavaksi 17. helmikuuta 2015 [24]. HTTP/2:n määritelmä julkaistiin dokumentilla RFC-7540 toukokuussa 2015 [25]. Standardisointiponnisteluja tukivat Chrome, Opera, Firefox, Internet Explorer 11, Safari, Amazon Silk ja Edge -verkkoselaimet [26]. Useimmat suuret selaimet olivat lisänneet HTTP/2 tuen vuoden 2015 loppuun mennessä [27]. W3Techs-sivuston mukaan maaliskuun 2019 jälkeen 33,9% maailman suosituimmista kymmenestä miljoonasta sivustosta tukivat HTTP/2-protokollaa [28].

HTTP/2 jättää suurimman osan HTTP/1.1:n korkean tason syntaksista ennalleen, kuten menetelmät, tilakoodit, otsikkokentät ja URI:t. Uutta on se, miten data kehystetään ja kuljetetaan asiakkaan ja palvelimen välillä [29]. Tehokkaat verkkosivustot minimoivat koko sivun esittämiseen vaadittavien pyyntöjen määrän kutistamalla resurssien määrää. Tämä toteutetaan vähentämällä koodin määrää ja pakkaamalla pienempiä kappaleita koodia yhteen.

HTTP/2 sallii palvelimen "työntää" sisältöä asiakkaalle. Palvelin voi siis lähettää resursseja ilman sovelluksen pyyntöä. Näin palvelin voi toimittaa sellaisia tietoja, joita se tietää sovelluksen tarvitsevan [30, 25]. Muut HTTP/2:n lisäparannukset tukevat pyyntöjen ja vastausten multipleksointia, otsikon pakkausta ja pyyntöjen priorisointia [31]. Koska HTTP/2 toimii kuitenkin vain yhden TCP-yhteyden päällä, on silti mahdollista, että esiintyy Head-Of-Line-Blocking -ongelmaa, jos TCP-paketit katoavat tai viivästyvät lähetyksessä [32]. HTTP/2 ei enää tue HTTP/1.1:n paloitellun datan koodausmekanismia, sillä se tarjoaa omat tehokkaammat mekanismit datan suoratoistoon [25].

3 TRANSPORT LAYER SECURITY

TLS (Transport Layer Security) on kryptografinen protokolla, joka suojaa tietoverkkojen dataliikennettä urkkimiselta. Sitä käytetään turvaamaan mm. viestintää ja verkkomaksuja, mutta myös tiedoston siirtäminen on salattavissa. Kyseessä on IETF:n standardiprotokolla, jonka tarkoitus on estää salakuuntelijoita sekä varmistaa viestien eheys ja muuttamattomuus. Sitä käytetään yleisimmin verkkoselainten, pikaviestimien, sähköpostin ja VoIP (Voice over IP) -yhteyksien salaamiseen.

TLS:n edeltäjä SSL (Secure Socket Layer) kehitettiin Netscapen toimesta vuonna 1995. Sen versiot 1.0 ja 2.0 sisälsivät niin suuria haavoittuvuuksia, että koko protokolla suunniteltiin uudelleen. Vuonna 1996 Netscape julkaisi version 3.0 protokollasta, joka toimi lähtökohtana TLS 1.0:lle.

TLS on tehokkaampi ja turvallisempi kuin SSL, sillä se omaa vahvemman viestin todennuksen, paremman avaimen generoinnin ja hienostuneemmat salausalgoritmit. TLS tukee ennalta jaettuja avaimia, elliptisen käyrän avaimia, turvallisia etäsalasanoja sekä Kerberos-protokollaa, jolla varmistetaan osapuolten identiteetti. SSL ei tue näistä ainuttakaan. TLS ja SSL eivät ole yhteensopivia, mutta TLS tarjoaa SSL-tuen vanhemmille laitteille, jotka eivät kykene käyttämään TLS:ää [33].

3.1 TLS:n perustekniikat

TLS käyttää sekä symmetristä että asymmetristä kryptografiaa, sillä tällä yhdistelmällä saadaan hyvä kompromissi suorituskyvyn sekä turvallisuuden suhteen. Symmetrisessä kryptografiassa data on salattu avaimella, jonka molemmat osapuolet tietävät. Tyypillisesti avain on pituudeltaan 128 bittiä, mutta nykypäivän prosessointiteholla 256 bittisiä tulisi käyttää. Symmetrinen salaus on tehokasta laskennallisesti, mutta se vaatii yhteisen avaimen jakamista molemmille osapuolille – salatusti. Tämä taas toteutetaan asymmetrisellä salauksella.

Asymmetrinen kryptografia käyttää avainpareja, eli julkisia (public) ja yksityisiä (private) avaimia (keys). Julkinen avain on matemaattisesti yhteyksissä yksityiseen avaimeen, mutta pituutensa vuoksi julkisesta avaimesta on lähes mahdotonta purkaa yksityistä avainta esille. Tällä tavoin viestin lähettäjä voi käyttää vastaanottajan julkista avainta salaukseen, jonka voi purkaa vain vastaanottajan yksityisellä avaimella.

Asymmetrisen kryptografian paras puoli on se, että se ei vaadi salattua yhteyttä avainten jakoon. Huono puoli on siinä, että avainten koko kasvaa nopeasti: Taulukosta 1 nähdään, kuinka symmetrisen avaimen koko kasvaa vain noin kolminkertaiseksi, mutta vastaavasti asymmetriset RSA sekä Diffie-Hellman -algoritmit kasvattavat avaimen 15-kertaisiksi. Elliptisen käyrän avainkoko kasvaa myös vain noin kolminkertaiseksi. Isompi avainkoko tarkoittaa myös parannettua sietokykyä avaimen murtamisyrityksiä vastaan.

***Taulukko 1**. Avainkokojen kasvu [34].*

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Avaimen koon kasvaessa yhteyden turvallisuus paranee, mutta prosessointiaika kasvaa räjähdysmäisesti. Tästä syystä TLS käyttää asymmetristä salausta vain istuntoavaimien (Session Key) vaihtoon, jonka jälkeen yhteys siirtyy käyttämään symmetristä salausta. Istunnon jälkeen yhteys suljetaan ja avaimet tuhotaan [35].

3.1.1 Salaussarja

Salaussarja (Cipher suite) on lista metodeista, joita käytetään salatun yhteyden luomiseen. Lista sisältää tiedon avaimenvaihtoon käytettävästä algoritmista, osapuolten vahvistamiseen käytettävästä algoritmista, itse salausalgoritmista sekä sanomien eheyden algoritmista [36]. Jokaiseen toimintoon käytettävistä algoritmeista löytyy useita vaihtoehtoja. Avaimenvaihtoon käytettyjä algoritmeja ovat esimerkiksi RSA, DH, ECDH sekä ECDHE. Osapuolten vahvistamiseen toimivat esimerkiksi RSA, DSA ja ECDSA. Itse datan salaamiseen voidaan käyttää AES-algoritmia eri bittiluvuilla. Sanomien eheyden varmistamiseen on ennen käytetty MD5- ja SHA-1-algorirmeja. Nykyään eheys tarkistetaan SHA256-algoritmilla.

Sovellus ja palvelin sopivat yhteyden luontiin käytettävistä algoritmeista käsittelemällä listoja salaussarjoista. Esimerkkinä tällaisesta salaussarjasta toimii rivi *TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384*.

Kryptiseltä näyttävä rivi voidaan purkaa osiin:

- *TLS* on käytettävä protokolla.
- *ECDHE* ilmoittaa käytettävästä avaintenvaihtoprotokollasta.
- *ECDSA* kertoo käytettävästä todennusalgoritmista.
- *WITH* on täytesana helpottamaan salaussarjan lukua.
- *AES_256_CBC* on datan salaukseen käytettävä algoritmi. Tässä esimerkissä kyseessä on AES256-CBC (Advanced Encryption Standard) 256-bittisellä salausavaimella ja CBC:llä, eli ketjutetulla lohkosalauksella [37].
- *SHA384* kuvaa käytettävää algoritmia sanomien eheydelle. SHA384 on tietty versio SHA -tiivistealgoritmista [38].

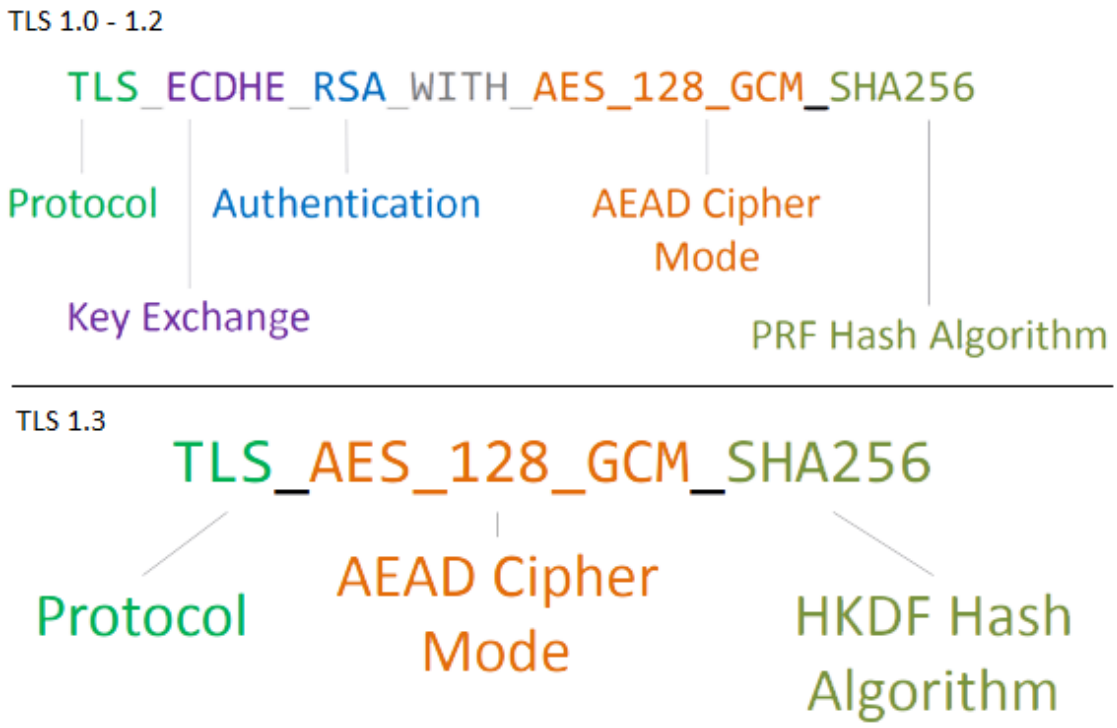
Joskus protokollaa ei tarvitse edes ilmoittaa, vaan voidaan määritellä pelkästään tärkeimmät algoritmit esimerkiksi seuraavasti *DHE_RSA_AES256_SHA*:

- *DHE* on avaintenvaihtoprotokolla.
- *RSA* on todennusalgoritmi.
- *AES256* on datan salausta varten.
- *SHA* on sanomien eheyteen käytettävä algoritmi.

Joskus algoritmit ovat implikoituja. Esimerkiksi sarjassa *RSA_AES256_SHA* varmistusalgoritmin voidaan olettaa olevan RSA.

Mikäli yhteyden osapuolet eivät pysty sopimaan käytettävistä algoritmeista, yhteyttä ei voida muodostaa. Algoritmeista sopiminen tapahtuu TCP-kättelyn jälkeen TLS-kättelyprotokollan yhteydessä [36].

TLS 1.3 muuttaa salaussarjojen käsittelyä hieman. Aiemmin salatun datan on pitänyt läpäistä vain eheyden tarkistukset, mutta uudessa versiossa kaiken datan pitää läpäistä myös todennus, jotta leikkaa-ja-liimaa -hyökkäykset eivät onnistu. Kyseessä on AEAD (Authenticated Encryption with Associated Data), joka varmistaa pakettien olevan samasta kontekstista. Jos pakettien konteksti on eri, ne hylätään [39].



Kuva 2. *Vertailu protokollaversioiden salaussarjojen käsittelystä [40].*

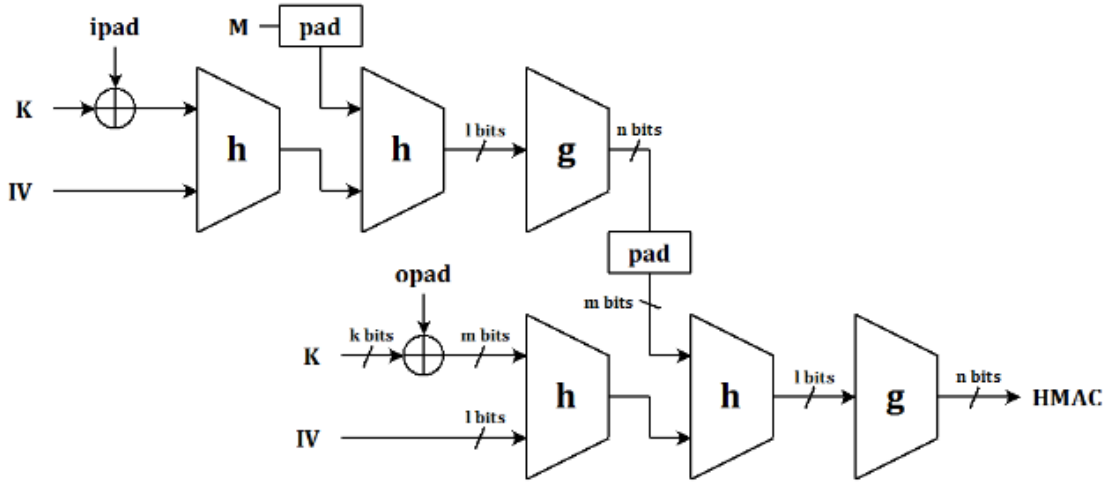
Kuvasta 2 nähdään erot protokollien salaussarjojen määrittelyn välillä. Aiemmat salaussarjat erittelivät käytettävän avaimenvaihtoprotokollan, todennusalgoritmin, salausalgoritmin sekä tiivistealgoritmin. Nyt avaimenvaihtoprotokolla sekä todennusalgoritmi sovitaan erikseen kättelyn yhteydessä [41].

3.1.2 Viestin todennus

Viestintodennuskoodi (Message Authentication Code, MAC) on lyhyt kappale informaatiota, jota käytetään viestin todentamiseen sekä eheyden ja aitouden varmistamiseen. Eheyden varmistamisella havaitaan tahattomat ja tahalliset muutokset viestin sisällössä, kun taas aitouden varmistuksella pyritään vahvistamaan viestin alkuperäinen lähettäjä. Yksinkertaisin tapa eheyden vahvistamiseen on laskea viestin tarkistussumma (checksum) käyttäen esimerkiksi CRC-algoritmia. Tarkistussumma voidaan liittää lähetettyyn viestiin, jonka vastaanottaja voi tarkistaa vastaanotettuaan viestin.

Tämän menetelmän ensisijainen haitta on suojauksen puute sanomien sisällön tahallisia muutoksia vastaan. Hyökkääjä voi muuttaa viestin sisältöä, laskea uuden tarkistussumman ja lopulta korvata alkuperäisen viestin omallaan. Tavallinen CRC-algoritmi auttaa havaitsemaan satunnaisesti muuttuneet viestien osat, muttei tahallisesti muutettuja viestejä.

TLS-protokolla käyttää viestintodennukseen HMAC-algoritmia. Kuva 3 esittää, kuinka viestistä laskettu tarkistussumma salataan, jolloin alkuperäisen viestin muuttaminen vaatisi tarkistussumman uudelleenlaskemista sekä sen salaamista samalla avaimella, kuin mitä salatun yhteyden yhteydessä käytetään. HMAC-laskennassa voidaan käyttää mitä tahansa kryptografista hajautusfunktioita (hash function), esimerkiksi MD5 tai SHA-1. Salauslujuus riippuu käytetyn hajautusalgoritmin vahvuudesta, tarkistussummien pituudesta sekä käytetyn salausavaimen pituudesta. HMAC-algoritmin tuottamat tarkistussummat ovat yksisuuntaisia, eikä niistä voida selvittää alkuperäistä viestiä.



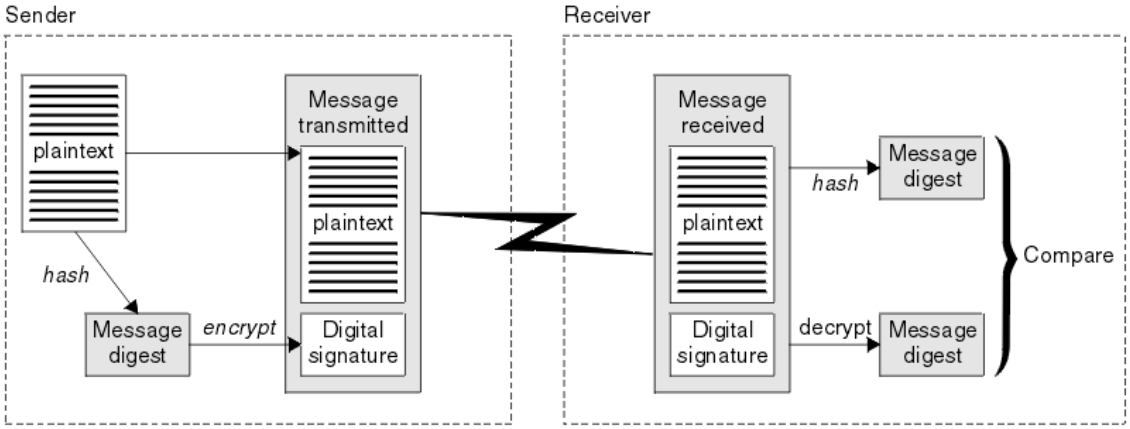
***Kuva 3**. Viestin todennus kuvattuna [42].*

Salaisen avaimen muuttamiseen voidaan käyttää muuttujia *ipad* ja *opad*. Niiden arvoiksi suositellaan sellaisia, jotka kummatkin muuttavat tulosteen *h* arvot erilaisiksi toisiinsa verrattuna. Käytännössä tämä tarkoittaa sitä, että salaista avainta muokataan kahdella eri tavalla. Viimeinen toiminto ottaa aiemmat *h*-arvot kahdesta sisääntulosta ja antaa tulokseksi yhden varmistukseen käytettävän koodin [42].

3.1.3 Digitaaliset allekirjoitukset

Digitaalinen allekirjoitus (Digital Signature) on prosessi, joka takaa, että viestin sisältöä ei ole muutettu siirron yhteydessä. Vastaanottaja voi yhä lukea viestin, mutta prosessi liittää mukaan digitaalisen allekirjoituksen, jonka voi purkaa vain lähettäjän julkisella avaimella. Tällä tavoin vastaanottaja voi varmistua siitä, että lähettäjä ja viestin sisältö ovat pysyneet ennallaan. Digitaalinen allekirjoitus luodaan ottamalla viestistä tiiviste ja salaamalla se allekirjoittajan yksityisellä avaimella. Asymmetrisen avainparin vuoksi menetelmä on hidas, joten on viisaampaa salata viestin tiiviste kuin itse viesti.

Digitaaliset allekirjoitukset vaihtelevat sen mukaan, mitä tietoja allekirjoitetaan, toisin kuin käsinkirjoitetut allekirjoitukset, jotka eivät riipu tietojen sisällöstä. Kaksi erilaista tietoa luo kaksi erilaista allekirjoitusta, vaikka allekirjoittaja olisikin sama. Molemmat allekirjoitukset voidaan tästä huolimatta tarkastaa samalla avaimella, eli viestin allekirjoittajan julkisella avaimella. Digitaalinen allekirjoitus ja varmistus tapahtuvat Kuvan 4 esittämällä tavalla:



Kuva 4. *Digitaalinen allekirjoitus ja sen varmistus.*

1 Lähettäjä laskee viestin tiivisteen ja salaa sen yksityisellä avaimellaan muodostaen digitaalisen allekirjoituksen.
2 Digitaalinen allekirjoitus lähetetään viestin mukana vastaanottajalle.
3 Vastaanottaja purkaa digitaalisen allekirjoituksen lähettäjän julkisella avaimella ja saa viestistä generoidun tiivisteen.
4 Vastaanottaja laskee viestistä tiivisteen itse ja vertaa digitaalisen allekirjoituksen mukana tullutta tiivistettä.
5 Mikäli tiivisteet ovat samat, ovat myös viestin sisältö ja lähettäjä säilyneet muuttumattomina.

Digitaaliset allekirjoitukset ovat osa eheyden ja todentamisen prosesseja. Samalla voidaan varmistua viestin alkuperästä, sillä vain oikea lähettäjä tuntee asymmetrisen avainparinsa molemmat avaimet [43].

3.1.4 Symmetrisen avaimen generointi

Symmetristen ja asymmetristen avainten generoiminen on matemaattinen prosessi, johon kumpikin osapuoli osallistuu. Kättelyprosessin yhteydessä sovellus ja palvelin sopivat yksityisen ja julkisen avainparin luomiseen käytettävästä algoritmista, avainten vaihtoon käytettävästä algoritmista sekä siitä, käytetäänkö uusiin yhteyksiin sertifikaateissa olevia julkisia avaimia vai luodaanko yhteyttä varten uudet avainparit.

Kättelyn yhteydessä osapuolet voivat sopia esimerkiksi käyttävänsä elliptisen käyrän x25519 metodia, Diffie-Hellman -metodia avainten vaihtoon sekä Ephemeral-avaimia. Kumpikin osapuoli siis generoi avaimensa käyttäen elliptistä käyrää, joiden avulla toteutetaan Diffie-Hellman-algoritmin mukainen symmetrisen avaimen generointi. Ephemeral tarkoittaa sitä, että uusi avainpari generoidaan jokaiselle yhteydelle sen sijaan, että käytettäisiin sertifikaatin julkista ja yksityistä avainta.

Yhteyttä avattaessa sovellus lähettää palvelimelle Client Hello -viestin. Viesti sisältää tiedot protokollan versiosta, satunnaista dataa avaimenluontiprosessia varten, mahdollisen aiemman istunnon ID:n, listan tuetuista salauksista, pakkausmetodeista ja laajennuksista. Palvelin vastaa omalla Server Hello -viestillään, joka sisältää samat tiedot palvelimen ominaisuuksista. Seuraavaksi palvelin lähettää sovellukselle oman sertifikaattinsa sekä julkisen avaimensa. TLS 1.2 -versiosta lähtien asymmetristen avainten generointiin on käytetty muun muassa elliptisen käyrän algoritmia x25519. Yksityinen avain, joka jää vain palvelimelle, generoidaan ottamalla 32 bittiä satunnaista dataa erilaisista lähteistä. Tämä 32-bittinen data kerrotaan arvolla, joka saadaan käyttämällä jotain X-akselin pistettä elliptiseltä käyrältä. Lopputuloksena saadaan yksityinen 32-tavuinen avain sekä elliptisen käyrän avulla generoitu julkinen 32-tavuinen avain. Palvelin lähettää julkisen avaimensa sovellukselle ja jää odottamaan prosessin jatkoa, johon tarvitaan sovelluksen tietoja.

Sovellus toteuttaa oman puolensa kuten palvelin. Ensiksi generoidaan 32 tavua satunnaista dataa, josta tulee yksityinen 32-tavuinen avain. Yksityisestä avaimesta generoidaan julkinen avain, joka lähetetään palvelimelle. Tässä kohtaa kummallakin osapuolella on neljä asiaa tiedossa, joita käytetään symmetrisen avaimen generointiin:

- Server Random sekä Client Random (satunnainen data, joka jaettiin Hello -viesteissä)
- toisen osapuolen julkinen avain
- oma yksityinen avain.

Seuraavassa vaiheessa osapuolet kertovat keskenään oman yksityisen avaimen ja vastapuolen julkisen avaimen. Matemaattisesti Diffie-Hellman-prosessi näyttää seuraavalta:

- Sovellus määrittelee kaksi alkulukua, *g* ja *p*, ja lähettää nämä palvelimelle
- Sovellus valitsee salaisen luvun *a*, muttei jaa sitä ulkopuolisille. Sen sijaan sovellus laskee lukujen *g*, *a* ja *p* avulla tuloksen *A* kaavalla

$$g^a \ mod \ p = A$$

ja lähettää sen palvelimelle.

- Palvelin tekee saman, mutta omalla salaisella luvullaan *b*:

$$g^b \ mod \ p = B$$

ja lähettää tuloksen *B* sovellukselle.

- Seuraavaksi molemmat osapuolet toistavat saman operaation käyttäen vastaanottamaansa lukua:

$$B^a \ mod \ p = g^{ab} \ mod \ p$$
$$A^b \ mod \ p = g^{ba} \ mod \ p$$

Lopputuloksena molemmilla osapuolilla on sama avain.

Tuloksena on 32-tavuinen avain, jota kutsutaan nimellä PreMasterSecret. PreMasterSecret-avaimesta generoidaan 48-tavuinen MasterSecret-avain lisäämällä yksityiseen avaimeen Client Random sekä Server Random -arvot. Seed-muuttujan *"master secret"* on generoitu avaimenvaihtoalgoritmin tilasta käyttämällä HMAC-versioita MD5- ja SHA-1 -tiivistefunktioista. Ohjelma 1 esittää pseudokoodilla, kuinka Client Random, Server Random sekä *"Master Secret"* yhdistetään ja niistä muodostetaan lopullinen MasterSecret-arvo [44, 45].

```
seed = "master secret" + client_random + server_random
a0 = seed
a1 = HMAC-SHA256(key=PreMasterSecret, data=a0)
a2 = HMAC-SHA256(key=PreMasterSecret, data=a1)
p1 = HMAC-SHA256(key=PreMasterSecret, data=a1 + seed)
p2 = HMAC-SHA256(key=PreMasterSecret, data=a2 + seed)
MasterSecret = p1[all 32 bytes] + p2[first 16 bytes]
```

***Ohjelma 1**: Master Secret -arvon laskeminen palvelimen toimesta*

MasterSecret-arvon avulla generoidaan loput tarvittavat avaimet, joilla salataan sekä vahvistetaan viestit: Client MAC Key, Server MAC Key, Client Write Key, Server Write Key, Client Write IV sekä Server Write IV. Seed-muuttujan key expansion -arvo on riippuvainen avainaikataulusta (key schedule), joka luodaan algoritmin sisäisesti [46]. Ohjelma 2:n pseudokoodi esittää, kuinka MasterSecret-arvoa käytetään luomaan tarvitut avaimet.

```
seed = "key expansion" + server_random + client_random
a0 = seed
a1 = HMAC-SHA256(key=MasterSecret, data=a0)
a2 = HMAC-SHA256(key=MasterSecret, data=a1)
a3 = HMAC-SHA256(key=MasterSecret, data=a2)
a4 = ...
p1 = HMAC-SHA256(key=MasterSecret, data=a1 + seed)
p2 = HMAC-SHA256(key=MasterSecret, data=a2 + seed)
p3 = HMAC-SHA256(key=MasterSecret, data=a3 + seed)
p4 = ...
p = p1 + p2 + p3 + p4 ...

client write mac key = [first 20 bytes of p]
server write mac key = [next 20 bytes of p]
client write key = [next 16 bytes of p]
server write key = [next 16 bytes of p]
client write IV = [next 16 bytes of p]
server write IV = [next 16 bytes of p]
```

***Ohjelma 2**: Avainten laskeminen Master Secret -arvosta*

Osapuolet tuottavat koko kättelystä tiivisteen käyttämällä MAC-avaimia, salaavat sen Write Key -avaimillaan ja lähettävät salatun tiivisteen FINISHED-viestin mukana toiselle osapuolelle. Vastaanottaja tuottaa samasta kättelystä oman tiivisteen ja tarkastaa sen olevan sama, kuin juuri vastaanottamansa. Tällä tavoin voidaan varmistua siitä, että kättely on säilynyt koskemattomana ja juuri luotuja symmetrisiä avaimia voidaan käyttää yhteyden salaamiseen [47, 48, 49, 43, 50].

3.1.5 TLS:n yleiset ominaisuudet

TLS-protokolla koostuu kahdesta pääkomponentista, jotka ovat kättelyprotokolla ja rekisteriprotokolla. Kättelyprotokolla todentaa kommunikoivat osapuolet, neuvottelee yhteydessä käytettävät kryptografiset ominaisuudet ja parametrit sekä muodostaa tarvittavan materiaalin avainten luontiin. Avainmateriaali tuotetaan luomalla salaiseksi jääviä tavusarjoja, joista algoritmit kokoavat tarvittavat sarjat avaimen luontiin. Osapuolet käyttävät asymmetristä salausta, kunnes avainmateriaali on siirretty kummallekin osapuolelle ja symmetrinen avain voidaan generoida [51]. Rekisteriprotokollan tehtävä on käsitellä ja salata siirrettävä data oikein. Rekisteriprotokolla jakaa uloslähtevän datan käsiteltäviin paloihin, järjestää ja käsittelee saapuvan datan, pakkaa uloslähtevät palat sekä purkaa saapuvan datan, mikäli saapuva data oli pakattua. Rekisteriprotokollan tehtäviin kuuluu myös sanoman eheyden tarkistamiseen käytettävät toiminnot, kuten pakettien tiivisteen luominen ja salaaminen sekä saapuvien pakettien tiivisteen tarkastaminen. Rekisteriprotokollan tärkein tehtävä on kuitenkin uloslähtevien pakettien

salaaminen kättelyprotokollassa sovitulla symmetrisellä avaimella sekä saapuvien pakettien purkaminen samaisella avaimella [52]. Protokolla sisältää myös varoitusprotokollan (Alert Protocol), jonka tehtävänä on varoittaa virheistä ja katkaista yhteys tarpeen tullen, mikäli jompikumpi osapuolista kokee virheen laadun tarpeeksi vakavaksi. Osa virheistä on asetettu niin vakaviksi, että kyseinen yhteys katkaistaan välittömästi mitätöimällä istunnon tunniste (SessionID) [53]. Varoitusprotokollan viestit ovat rekisteriprotokollan suojaamia, mikä tarkoittaa sitä, että varoitusviestit eivät vuoda ulkopuolisille [54].

TLS on luotu yksinkertaisten filosofioiden pohjalta:

1. Kryptografinen varmuus, eli TLS:ää tulisi käyttää turvallisen yhteyden muodostamiseen kahden osapuolen välille.
2. Yhteensopivuus. Riippumattomien ohjelmoijien tulisi pystyä kehittämään sovelluksia, jotka voivat TLS:n avulla vaihtaa onnistuneesti salausparametrejä tuntematta toisen koodia.
3. Laajennettavuus. TLS pyrkii tarjoamaan kehyksen, johon uudet julkiset avaimet ja massasalausmetodit voidaan sisällyttää tarpeen mukaan. Näin saavutetaan myös kaksi alakohtaista tavoitetta: estetään tarve luoda uusi protokolla, eikä riskeerata uusien haavoittuvuuksien syntyä. Samalla kertaa pyritään välttämään kokonaisen uuden salaussarjan käyttöönotto, joka saattaisi sisältää haavoittuvuuksia.
4. Tehokkuus. Salaustoiminnot ovat yleensä intensiivisiä ja vaativat suuria laskutoimituksia erityisesti julkisen avaimen operaatioissa. Tästä syystä TLS-protokollaan on sisällytetty valinnainen välimuisti istuntoja varten (Session Caching Scheme), jonka tarkoituksena on vähentää yhteyksien uudelleenluomista. Tällä myös vähennetään verkkoaktiviteettia [55].

Näiden kulmakivien ansiosta TLS on päätynyt alan standardiksi, jota kehitetään jatkuvasti.

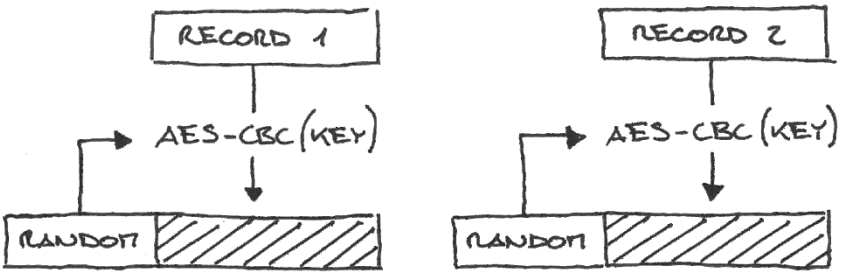
3.2 TLS-protokollan versiot

TLS 1.0 (RFC-2246) kehitettiin Christopher Allenin ja Tim Dierksin toimesta korvaamaan SSL 3 [56]. TLS on pohjimmiltaan SSL 3.1+, tai SSLv4. Nimenvaihdoksen pohjalla on sopimus Microsoftin ja Netscapen välillä, jossa kumpikin osapuoli hyväksyi sen, että IETF ottaisi protokollan haltuunsa. Yksi Microsoftin vaatimuksista oli protokollan nimenmuutos, jottei tilanne vaikuttaisi siltä, että IETF hyväksyi vain Netscapen puolen sopimuksesta – SSL oli kuitenkin Netscapen luomus [57].

3.2.1 TLS 1.1

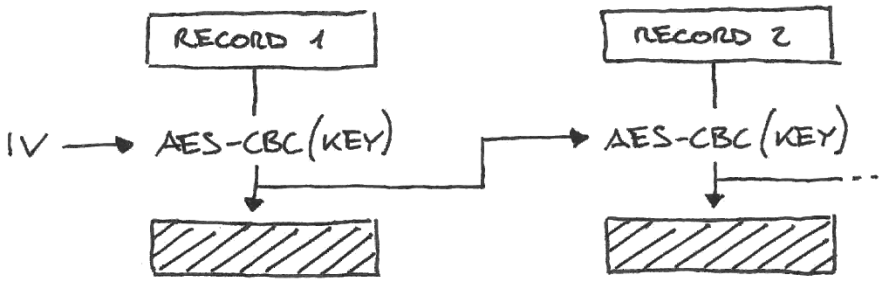
Vuonna 1999 julkaistu TLS 1.0 versio sai päivityksen vuonna 2006, kun protokollaa vahvistettiin lisäämällä suojauksia ketjusalakirjoituksiin keskittyviä hyökkäyksiä vastaan. Protokollan versionumeroksi tuli 1.1, ja se julkaistiin dokumenttinumerolla RFC-4346 [58]. Nimekkäin näistä hyökkäyksistä oli BEAST (Browser Exploit Against SSL/TLS). Hyökkäys hyväksikäytti lohkosalauksessa ollutta haavoittuvuutta [59].

Implisiittinen alustusvektori (implicit initialization vector, IV) vaihdettiin eksplisiittiseen alustusvektoriin (explicit IV). Alustusvektori on ennustamaton lukusarja, jota käytetään salausalgoritmissa ensimmäisenä lukuarvona [60, 61]. Näiden vektorien ero on se, että implisiittinen alustusvektori luotiin aiemman lohkon pohjalta, kun taas eksplisiittinen vektori luodaan riippumattomien lähteiden avulla ja lähetetään aiemman paketin mukana [59].



Kuva 5. *Eksplisiittisen alustusvektorin hyödyntäminen [62].*

Kuvasta 5 nähdään, kuinka lohkojen salaus tapahtuu eksplisiittisillä alustusvektoreilla. Alustusvektori on luotu erikseen jokaista lohkoa kohden. Implisiittinen alustusvektori luotiin käyttämällä edellisen lohkon tietoja. Tämä prosessi on esitetty Kuvassa 6.



Kuva 6. *Implisiittisen alustusvektorin käyttö [62].*

Alustusvektorin käsittelyn lisäksi täytevirheiden (padding errors) käsittelyä muutettiin ja sertifikaattien parametrit saatiin rekisteröityä IANA:n alaisuudessa. Tämä paransi luottojärjestelmää, sillä yhteyden osapuolet saatiin vahvistettua kolmannen osapuolen avulla [63].

TLS 1.1 käyttää samaa listaa salaukseen käytettävistä metodeista, kuin TLS 1.0 [64]. Listaa on jatkettu useassa eri RFC-dokumentissa aina, kun protokollaan on tuotu uusi sarja algoritmeja (ks. Taulukko 2). Alkuperäisiä yhdistelmiä on 28.

Taulukko 2. *Lista TLS 1.0 – 1.1 versioiden tukemista salaussarjoista [65].*

TLS v1.0 - 1.1 cipher suites	
Peruskirjastot	
12	TLS_DH_anon_WITH_3DES_EDE_CBC_SHA
	TLS_DH_anon_WITH_RC4_128_MD5
	TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
	TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
	TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
	TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
	TLS_RSA_WITH_3DES_EDE_CBC_SHA
	TLS_RSA_WITH_IDEA_CBC_SHA
	TLS_RSA_WITH_NULL_MD5
	TLS_RSA_WITH_NULL_SHA
	TLS_RSA_WITH_RC4_128_MD5
	TLS_RSA_WITH_RC4_128_SHA
Export 1024	
1	TLS_DHE_DSS_WITH_RC4_128_SHA
Camellia - RFC-4132	
12	TLS_DH_anon_WITH_CAMELLIA_128_CBC_SHA
	TLS_DH_anon_WITH_CAMELLIA_256_CBC_SHA
	TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA
	TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA
	TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA
	TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA
	TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA
	TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA
	TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
	TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
	TLS_RSA_WITH_CAMELLIA_128_CBC_SHA
	TLS_RSA_WITH_CAMELLIA_256_CBC_SHA
GOST - draft-chudov-cryptopro-cptls	
4	TLS_GOSTR341001_WITH_28147_CNT_IMIT
	TLS_GOSTR341001_WITH_NULL_GOSTR3411
	TLS_GOSTR341094_WITH_28147_CNT_IMIT
	TLS_GOSTR341094_WITH_NULL_GOSTR3411

Elliptisen käyrän kirjastot	
15	TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
	TLS_ECDH_anon_WITH_AES_128_CBC_SHA
	TLS_ECDH_anon_WITH_AES_256_CBC_SHA
	TLS_ECDH_anon_WITH_NULL_SHA
	TLS_ECDH_anon_WITH_RC4_128_SHA
	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
	TLS_ECDHE_ECDSA_WITH_NULL_SHA
	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
	TLS_ECDHE_RSA_WITH_NULL_SHA
	TLS_ECDHE_RSA_WITH_RC4_128_SHA
AES - RFC-3268	
12	TLS_DH_anon_WITH_AES_128_CBC_SHA
	TLS_DH_anon_WITH_AES_256_CBC_SHA
	TLS_DH_DSS_WITH_AES_128_CBC_SHA
	TLS_DH_DSS_WITH_AES_256_CBC_SHA
	TLS_DH_RSA_WITH_AES_128_CBC_SHA
	TLS_DH_RSA_WITH_AES_256_CBC_SHA
	TLS_DHE_DSS_WITH_AES_128_CBC_SHA
	TLS_DHE_DSS_WITH_AES_256_CBC_SHA
	TLS_DHE_RSA_WITH_AES_128_CBC_SHA
	TLS_DHE_RSA_WITH_AES_256_CBC_SHA
	TLS_RSA_WITH_AES_128_CBC_SHA
	TLS_RSA_WITH_AES_256_CBC_SHA
SEED - RFC-4162	
6	TLS_DH_anon_WITH_SEED_CBC_SHA
	TLS_DH_DSS_WITH_SEED_CBC_SHA
	TLS_DH_RSA_WITH_SEED_CBC_SHA
	TLS_DHE_DSS_WITH_SEED_CBC_SHA
	TLS_DHE_RSA_WITH_SEED_CBC_SHA
	TLS_RSA_WITH_SEED_CBC_SHA

Taulukosta 2 nähdään, kuinka listaa on jatkettu kolmella RFC-dokumentilla sekä yhdellä luonnoksella, *draft-chudov-cryptopro-cptls*. Mikäli jossain näistä 62:sta yhdistelmässä olisi haavoittuvuus, sen löytäminen ajoissa olisi haastavaa.

3.2.2 TLS 1.2

TLS 1.2 julkaistiin vuonna 2008 dokumenttinumerolla RFC-5246. Päivityksen tärkein päämäärä oli poistaa MD5 ja SHA-1 -tiivistealgoritmeihin liittyvät riippuvuudet. Aiempaan versioon verrattuna uusi versio oli samalla joustavampi. Muutoksien määrä oli merkittävä.

MD5/SHA-1 -yhdistelmän käyttö näennäissatunnaisfunktioissa (Pseudorandom function, PRF) korvattiin salaussarjoissa määritellyillä näennäissatunnaisfunktioilla. Saman yhdistelmän käyttö digitaalisessa allekirjoituksessa korvattiin yksittäisellä tiivisteellä, ja allekirjoitusten yhteyteen lisättiin kenttä, josta käy ilmi käytetty tiivistealgoritmi.

Sovellus ja palvelin saivat paremmat ominaisuudet tiiviste- ja allekirjoitusalgoritmien sopimiseen ja prosessia selvennettiin. AES-salaussarja sekä TLS-laajennusten määritelmät lisättiin protokollaan, ja versionumeron tarkastusta tiukennettiin EncryptedPreMaster-avaimen kohdalla.

Datan varmistamiseen käytetyn muuttujan pituus vaihdettiin riippuvaiseksi salaussarjoista, ja *Bleichenbacher/Dlima*-hyökkäyksen vastaiset suojaustoimet kirjoitettiin selvemmäksi. Varoitusviestejä vaativia tilanteita lisättiin, ja sertifikaatin puuttuessa sovelluksen tulee lähettää tyhjä sertifikaattilista.

Salausalgoritmista *TLS_RSA_WITH_AES_128_CBC_SHA* tehtiin pakollinen kaikissa protokollan implementaatioissa, HMAC-SHA256 lisättiin salaussarjoihin ja samalla poistettiin IDEA sekä DES -salaussarjat, sillä ne eivät olleet enää turvallisia käyttää [66].

Päivityksistä huolimatta TLS 1.2 on saanut oman osansa haavoittuvuuksista. Pohjimmiltaan protokollassa ei ollut mitään vikaa vaan sen implementaatiossa. Nimekkäisiin haavoittuvuuksiin kuuluivat muun muassa *Heartbleed* [67], joka oli OpenSSL-ohjelmiston virhe, *BERserk* [68], joka koski lähinnä Google Chrome ja Mozilla Firefox -selaimia, sekä Apple-yhtiön implementaatiossa ollut *goto fail;* -haavoittuvuus [69]. Muita haavoittuvuuksia olivat POODLE [70] ja Lucky13, jotka hyväksikäyttivät salakirjoitusketjun (cipher block chain, CBC) haavoittuvuuksia, sekä LogJam, joka käytti yleisen lukukuntaseulan algoritmia murtaakseen 512 – 1024 bittisiä Diffie-Hellman avaimia [71, 72].

TLS 1.2 laajensi aiempaa listaa merkittävästi. Alkuperäisten yhdistelmien määrä nousi 28:sta aina 105 asti, 3DES-algoritmia käyttävät yhdistelmät poistettiin täysin ja AES-algoritmi lisättiin perussarjoihin. Taulukosta 3 käy ilmi yhdistelmien määrä. Alkuperäisiä yhdistelmiä on 105 ja lisäykset ovat tuoneet 27 yhdistelmää lisää. Listaa pidettiin turvallisena, mutta IETF toivoi pääsevänsä eroon ylimääräisistä, sillä osa yhdistelmistä oli helppo määritellä väärin

Taulukko 3. *Tuetut salaussarjat TLS 1.2 -versiossa [65].*

TLS v1.2 cipher suites	
Peruskirjastot	
45	DHE_RSA_WITH_AES_128_CCM
	DHE_RSA_WITH_AES_128_CCM_8
	DHE_RSA_WITH_AES_256_CCM
	DHE_RSA_WITH_AES_256_CCM_8
	ECDHE_ECDSA_WITH_AES_128_CCM
	ECDHE_ECDSA_WITH_AES_128_CCM_8
	ECDHE_ECDSA_WITH_AES_256_CCM
	ECDHE_ECDSA_WITH_AES_256_CCM_8
	RSA_WITH_AES_128_CCM
	RSA_WITH_AES_128_CCM_8
	RSA_WITH_AES_256_CCM
	RSA_WITH_AES_256_CCM_8
	TLS_DH_anon_WITH_AES_128_CBC_SHA256
	TLS_DH_anon_WITH_AES_128_GCM_SHA256
	TLS_DH_anon_WITH_AES_256_CBC_SHA256
	TLS_DH_anon_WITH_AES_256_GCM_SHA384
	TLS_DH_DSS_WITH_AES_128_CBC_SHA256
	TLS_DH_DSS_WITH_AES_128_GCM_SHA256
	TLS_DH_DSS_WITH_AES_256_CBC_SHA256
	TLS_DH_DSS_WITH_AES_256_GCM_SHA384
	TLS_DH_RSA_WITH_AES_128_CBC_SHA256
	TLS_DH_RSA_WITH_AES_128_GCM_SHA256
	TLS_DH_RSA_WITH_AES_256_CBC_SHA256
	TLS_DH_RSA_WITH_AES_256_GCM_SHA384
	TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
	TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
	TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
	TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
	TLS_RSA_WITH_AES_128_CBC_SHA256
	TLS_RSA_WITH_AES_128_GCM_SHA256
	TLS_RSA_WITH_AES_256_CBC_SHA256
	TLS_RSA_WITH_AES_256_GCM_SHA384
	TLS_RSA_WITH_NULL_SHA256
ARIA - RFC-6209	
16	TLS_DHE_DSS_WITH_ARIA_128_GCM_SHA256
	TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384
	TLS_DHE_PSK_WITH_ARIA_128_GCM_SHA256
	TLS_DHE_PSK_WITH_ARIA_256_GCM_SHA384
	TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256
	TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384
	TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256
	TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384
	TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256
	TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384
	TLS_PSK_WITH_ARIA_128_GCM_SHA256
	TLS_PSK_WITH_ARIA_256_GCM_SHA384
	TLS_RSA_PSK_WITH_ARIA_128_GCM_SHA256
	TLS_RSA_PSK_WITH_ARIA_256_GCM_SHA384
	TLS_RSA_WITH_ARIA_128_GCM_SHA256
	TLS_RSA_WITH_ARIA_256_GCM_SHA384
Camellia HMAC-Based - RFC-6367	
4	TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
	TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384
	TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
	TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384
Ennakkoon jaettujen avainten (PSK) kirjastot	
60	DHE_PSK_WITH_3DES_EDE_CBC_SHA
	DHE_PSK_WITH_AES_128_CBC_SHA
	DHE_PSK_WITH_AES_128_CBC_SHA256
	DHE_PSK_WITH_AES_128_CCM
	DHE_PSK_WITH_AES_128_CCM_8
	DHE_PSK_WITH_AES_128_GCM_SHA256
	DHE_PSK_WITH_AES_256_CBC_SHA
	DHE_PSK_WITH_AES_256_CBC_SHA384
	DHE_PSK_WITH_AES_256_CCM
	DHE_PSK_WITH_AES_256_CCM_8
	DHE_PSK_WITH_AES_256_GCM_SHA384
	DHE_PSK_WITH_CAMELLIA_128_CBC_SHA256
	DHE_PSK_WITH_CAMELLIA_256_CBC_SHA384
	DHE_PSK_WITH_NULL_SHA
	DHE_PSK_WITH_NULL_SHA256
	DHE_PSK_WITH_NULL_SHA384
	DHE_PSK_WITH_RC4_128_SHA
	ECDHE_PSK_WITH_3DES_EDE_CBC_SHA
	ECDHE_PSK_WITH_AES_128_CBC_SHA
	ECDHE_PSK_WITH_AES_128_CBC_SHA256
	ECDHE_PSK_WITH_AES_256_CBC_SHA
	ECDHE_PSK_WITH_AES_256_CBC_SHA384
	ECDHE_PSK_WITH_CAMELLIA_128_CBC_SHA256
	ECDHE_PSK_WITH_CAMELLIA_256_CBC_SHA384
	ECDHE_PSK_WITH_NULL_SHA
	ECDHE_PSK_WITH_NULL_SHA256
	ECDHE_PSK_WITH_NULL_SHA384
	ECDHE_PSK_WITH_RC4_128_SHA
	PSK_WITH_3DES_EDE_CBC_SHA
	PSK_WITH_AES_128_CBC_SHA
	PSK_WITH_AES_128_CBC_SHA256
	PSK_WITH_AES_128_CCM
	PSK_WITH_AES_128_CCM_8
	PSK_WITH_AES_128_GCM_SHA256
	PSK_WITH_AES_128_GCM_SHA256
	PSK_WITH_AES_256_CBC_SHA
	PSK_WITH_AES_256_CBC_SHA384
	PSK_WITH_AES_256_CCM
	PSK_WITH_AES_256_CCM_8
	PSK_WITH_AES_256_GCM_SHA384
	PSK_WITH_AES_256_GCM_SHA384
	PSK_WITH_CAMELLIA_128_CBC_SHA256
	PSK_WITH_CAMELLIA_256_CBC_SHA384
	PSK_WITH_NULL_SHA
	PSK_WITH_NULL_SHA256
	PSK_WITH_NULL_SHA384
	PSK_WITH_RC4_128_SHA
	RSA_PSK_WITH_3DES_EDE_CBC_SHA
	RSA_PSK_WITH_AES_128_CBC_SHA
	RSA_PSK_WITH_AES_128_CBC_SHA256
	RSA_PSK_WITH_AES_128_GCM_SHA256
	RSA_PSK_WITH_AES_256_CBC_SHA
	RSA_PSK_WITH_AES_256_CBC_SHA384
	RSA_PSK_WITH_AES_256_GCM_SHA384
	RSA_PSK_WITH_CAMELLIA_128_CBC_SHA256
	RSA_PSK_WITH_CAMELLIA_256_CBC_SHA384
	RSA_PSK_WITH_NULL_SHA
	RSA_PSK_WITH_NULL_SHA256
	RSA_PSK_WITH_NULL_SHA384
	RSA_PSK_WITH_RC4_128_SHA
ChaCha20-Poly1305 - RFC-7905	
7	TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256
	TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
	TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256
	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
	TLS_PSK_WITH_CHACHA20_POLY1305_SHA256
	TLS_RSA_PSK_WITH_CHACHA20_POLY1305_SHA256

3.2.3 TLS 1.3

TLS-protokolla on kärsinyt erilaisista ongelmista viime vuosien aikana. Se suunniteltiin insinöörien toimesta käyttäen matemaatikkojen työkaluja. Monet suunnittelupäätökset tehtiin heuristisella ja epätäydellisellä ymmärryksellä siitä, miten vankka salausprotokolla suunnitellaan. Siitä ei kuitenkaan voi syyttää protokollan suunnittelijoita, sillä olihan teknologia-alakin vielä alkutekijöissään tietoturvan suhteen [73].

TLS 1.2 -versioon tyytymätön IETF halusi eroon kahden RTT:n ajan vaativasta kättelystä ja muista versioon liittyvistä ongelmista. Vuoden 2013 elokuussa Eric Rescorla julkaisi toivelistan uutta protokollaversiota varten [73]. Kättelyn pituus haluttiin pudottaa kahden RTT:n kestosta yhteen RTT:hen, ja istunnon palautuminen haluttiin lyhentää yhdestä RTT:stä nollaan. Aiempi kättely paljasti osapuolten identiteetin sekä kaikki tuetut laajennukset. Tästä haluttiin eroon suojaamalla molempien osapuolien identiteetti passiivisilta hyökkääjiltä ja vähintään toisen osapuolen identiteetti aktiivisilta hyökkääjiltä. Laajennukset haluttiin myös salata ulkopuolisilta. Kaiken tämän haluttiin olevan yksi konfiguraatiovaihtoehdoista, eikä ainoa moodi, jota käyttää. Palvelimen avaimenvaihtoon käytetty allekirjoitus ei suojannut koko kättelyä, joka mahdollisti sovelluksen sekoittamisen. Tarkkaa suunnitelmaa tämän korjaamiselle ei ollut vaan ainoastaan lause "Tee tälle jotain" [74]. Saman listan mukaan IETF halusi eroon symmetrisistä salauksista, joista "olivat surullisia" oletettavasti salauksiin liittyvien heikkouksien vuoksi. Näihin kuului muun muassa Rivest Cipher 4 eli RC4 sekä Cipher Block Chaining, CBC [74].

Muutoslista saatiin tiivistettyä neljään pääpointtiin: kättelyn kestoa piti lyhentää, kättelyn tietoja tuli salata enemmän, protokollan piti olla vahvempi ristikkäisprotokollahyökkäyksiä vastaan ja vanhoista ominaisuuksista haluttiin eroon. Protokollamääritelmää työstettiin avoimena prosessina vapaaehtoisten toimesta neljän vuoden ajan [73].

Protokollan versio 1.3 viimeisteltiin loppukesästä vuonna 2018 ja se sai dokumenttinumerokseen RFC-8446 [2]. Versio paransi turvallisuutta ja nopeutta sekä lisäsi uusia ominaisuuksia. Protokollasta poistettiin kaikki vanhat salausalgoritmit, joten jäljelle jäivät vain AEAD-kykyiset algoritmit. AEAD (Authenticated Encryption with Associated Data) -algoritmit vaativat eheyden tarkistuksen lisäksi varmistuksen estäen leikkaa-ja-liimaa -hyökkäykset [75]. Zero-RTT (0-RTT) -ominaisuus lisättiin säästämään aikaa istuntojen uusinnassa, mutta tämä heikentää yhteyden suojauksen tasoa mahdollistaen toistohyökkäykset [76]. Staattiset RSA- ja Diffie-Hellman-salakirjoitukset poistettiin, ja kaikki julkisen avaimen avainvaihtomekaniikat tukevat jatkuvaa salassapitoa, vaikka aiemmat avaimet vuotaisivatkin. Kaikki kättelyyn liittyvät viestit

Server Hello -viestin jälkeen salataan, ja avaimien luontiin käytetyt funktiot ovat suunniteltu uudelleen käyttämään HMAC-pohjaista avaimenluontifunktiota HKDF (HMAC Key Derivation Function) lähtökohtana avaimille. Kättelyä suorittavat funktiot järjestettiin johdonmukaisesti, ja tarpeettomat viestit kättelyn suoritukseen liittyen poistettiin. Elliptisen käyrän kryptografia (ECC) asetettiin oletusarvoiseksi toimintatavaksi, ja sen mukana tuotiin uusia algoritmeja. Tiivistäminen, mukautetut Diffie-Hellman Ephemeral (DHE) -ryhmät sekä Digital Signature Algorithm (DSA) poistettiin, ja täytefunktio RSA Padding käyttää nyt todennäköisyyksiin perustuvaa allekirjoitusmenetelmää Probabilistic Signature Scheme (PSS) [77]. TLS 1.2 -protokollassa käytetty version varmistusmetodi korvattiin laajennuksiin liitettävällä versionumerolistalla. Istunnon uusimiseen käytetyt PSK-pohjaiset (Pre-Shared Key) salaussarjat korvattiin yhdellä uudella PSK:n vaihtometodilla [78, 51].

Vaikka protokollaa parannettiin merkittävästi turvallisuuden ja nopeuden puolesta, yksi mielenkiintoisimmista muutoksista oli Daniel J. Bernsteinin kirjoittamien algoritmien tuonti protokollaan [79]. Salaussarjojen määrää pienennettiin merkittävästi, mutta samalla kertaa lisättiin "tylsää kryptoa" [80]. IETF halusi eroon algoritmeista, jotka ovat olleet, tai yhä ovat, haavoittuvaisia. He turvautuivat djb:n algoritmeihin [79, 81]. Djb:n algoritmeihin kuuluvat ChaCha20, hiottu versio virtaussalausalgoritmista Salsa20, viestien todennukseen käytetty Poly1305 sekä Curve25519, joka on 128-bittinen elliptisen käyrän implementaatio Diffie-Hellman avainvaihtoa varten [82, 83, 84, 85].

TLS 1.3 -salaussarjat

TLS 1.3 poisti listalta kaikki yhdistelmät, joissa käytettiin algoritmia, johon IETF ei ollut tyytyväinen [74]. Näitä olivat esimerkiksi RC4, CBC, RSA, 3DES, Camellia, MD5 sekä SHA-1. Lopullinen lista sisälsi vain viisi algoritmiyhdistelmää, kuten Taulukko 4 osoittaa.

Taulukko 4. *Lista TLS 1.3 -tuetuista algoritmiyhdistelmistä [65].*

TLS v1.3 cipher suites	
Peruskirjastot	
5	TLS_AES_128_CCM_8_SHA256
	TLS_AES_128_CCM_SHA256
	TLS_AES_128_GCM_SHA256
	TLS_AES_256_GCM_SHA384
	TLS_CHACHA20_POLY1305_SHA256

Protokollan versio 1.3 on kuitenkin niin uusi, ettei uusia algoritmeja ole vielä lisätty. Protokollassa on osittainen tuki TLS 1.2 -ominaisuuksille, vaikka uusi versio onkin

suunniteltu tyhjästä. Tarkoituksena on päästä irti historian mukana tuomista ongelmista [79].

3.3 Eri TLS-versioiden kättelytavat

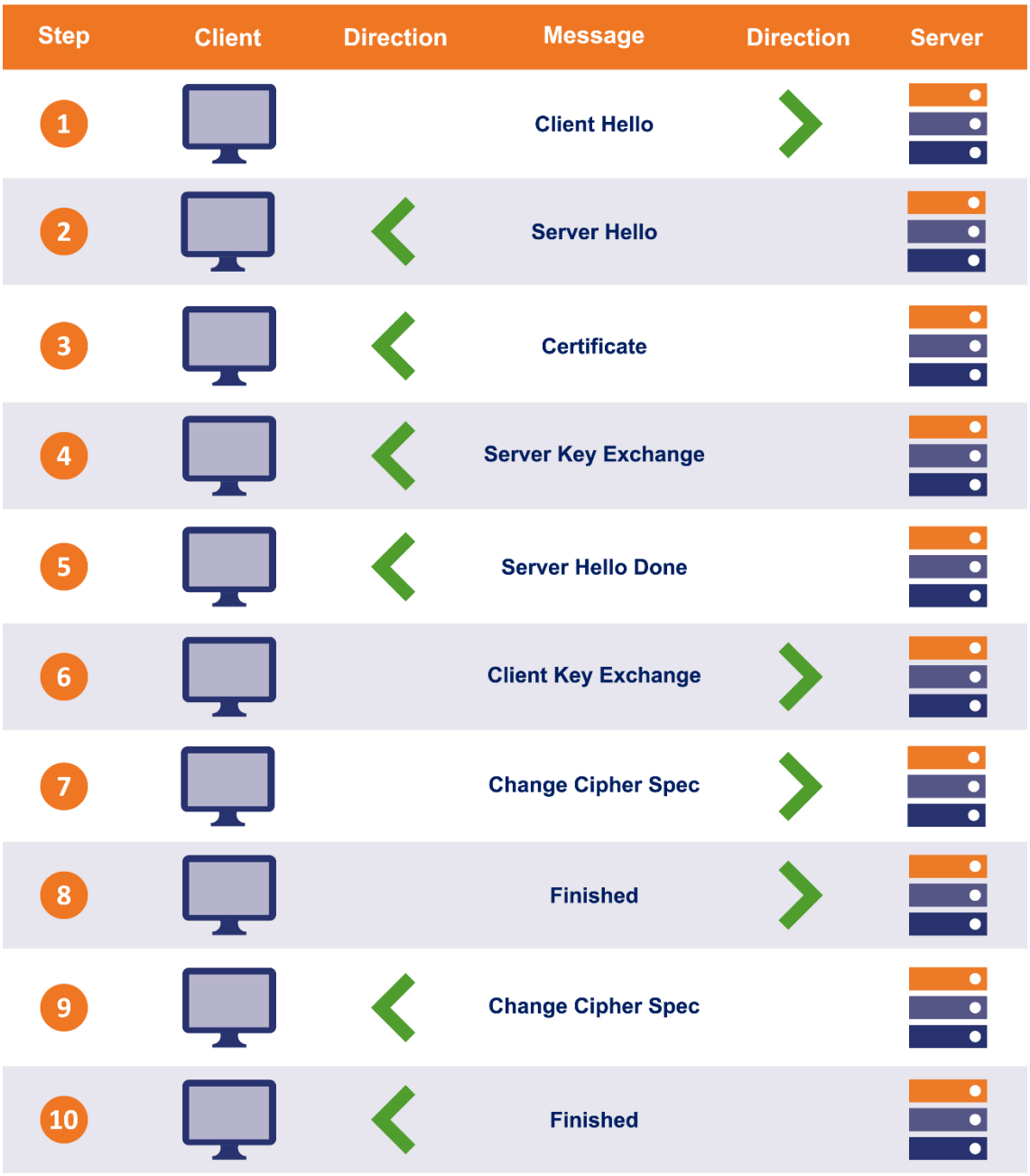
Kättely on prosessi, jonka avulla sovellus ja palvelin muodostavat salatun yhteyden. Ennen kuin sovellus ja palvelin voivat vaihtaa tietoa keskenään, tulee osapuolten neuvotella salatun yhteyden tiedoista. Molempien osapuolten tulee hyväksyä käytetyn TLS-protokollan versio sekä salaukseen käytettävät algoritmit ja varmistaa sertifikaattien oikeellisuus [86].

Sovelluksen on jaettava useita tietoja itsestään: versionumero tukemastaan TLS-protokollasta, 32-bittinen satunnaisluku symmetrisen avaimen muodostamiseen, istunnon tunnus (session ID), tiivistämismetodit pakettien koon pienentämiseen, toivottu salaussarja pakettien salaamiseen, vahvistamiseen sekä todentamiseen ja lista yhteydelle lisättävistä ominaisuuksista [87].

Tarvittavien tietojen vaihto tarkoittaa myös sitä, että salatun yhteyden avaamiseen kuluu aikaa useamman edestakaisen paketin siirron verran. Tämä lisää viivettä ja ylimääräistä liikennettä verkossa [86].

3.3.1 TLS 1.1 ja 1.2 -versioiden kättely

Kättelyn osuus salatun yhteyden luonnista on pysynyt pitkälti samanlaisena aina TLS 1.0 -versiosta lähtien aina TLS 1.2 -versioon asti. Sovellus on aina lähettänyt Hello-viestin. Palvelin on vastannut omalla Hello-viestillään, sertifikaatilla, avainvaihtomekaniikoilla ja Hello Done -viestillä. Sovellus on lähettänyt vastaukseksi oman osansa avaimenvalintamekaniikasta ja vaihtanut asymmetrisestä salauksesta symmetriseen salaukseen. Palvelin on vaihtanut salauksensa symmetriseksi ja vastannut Finished-viestillä [88].



Kuva 7. *Kättelyprosessi TLS 1.0 – 1.2 versioilla [88].*

Kuvasta 7 nähdään, kuinka edestakaisia viestinvaihtoja muodostuu kaksi. Kyseessä on RTT (Round-trip-time), joka vaikuttaa yhteyden luomisen nopeuteen. Ensimmäinen Client Hello -viesti, sisältää tiedon protokollan versionumerosta, 32 bittiä satunnaista dataa, mahdollisen istunnon tunnuksen (session ID), mikäli sellainen on ja listan tuetuista salausalgoritmeista, pakkausmetodesta sekä laajennuksista.

Client Hello-viesti voi näyttää esimerkiksi seuraavalta:

16 03 01 00 a5 01 00 00 a1 03 03 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 00 00 20 cc a8 cc a9 c0 2f c0 30 c0 2b c0

2c c0 13 c0 09 c0 14 c0 0a 00 9c 00 9d 00 2f 00 35 c0 12 00 0a 01 00 00 58 00 00 00 18 00 16 00 00 13 65 78 61 6d 70 6c 65 2e 75 6c 66 68 65 69 6d 2e 6e 65 74 00 05 00 05 01 00 00 00 00 00 0a 00 0a 00 08 00 1d 00 17 00 18 00 19 00 0b 00 02 01 00 00 0d 00 12 00 10 04 01 04 03 05 01 05 03 06 01 06 03 02 01 02 03 ff 01 00 01 00 00 12 00 00.

Ensimmäiset 5 heksadesimaalia, *16 03 01 00 a5*, määrittelevät rekisteriotsakkeen (Record Header) tietoja. Ensimmäinen näistä heksadesimaaleista, *16*, viittaa kättelyn rekisteriin. Seuraavat kaksi heksadesimaalia *03 01* määrittelevät protokollan version. Tässä esimerkissä versionumero on 3.1, joka tunnetaan myös TLS:n 1.0 versiona. Viimeiset kaksi heksadesimaalia, *00 a5*, ilmoittavat seuraavien 165 tavun olevan osa kättelyyn kuuluvaa viestiä. Heksadesimaali *a5* on 165 normaalissa kymmenjärjestelmässä.

Seuraavat neljä heksadesimaalia, *01 00 00 a1*, ovat kättelyn otsakkeita. Ensimmäinen heksadesimaali *01* kertoo viestin tyypin, joka tässä kohtaa kättelyä on Client Hello. Loput kolme heksadesimaalia kertovat, montako tavua Client Hello -viestiä vielä seuraa.

Kaksi seuraavaa numeroa kertovat sovelluksen protokollan versionumerosta. Esimerkin *03 03* viittaa protokollan versioon 3.3, eli TLS 1.2. Transport Layer Security -protokollaa pidetään SSL3-protokollan seuraajana, eli versionumero 3.1 viittaisi TLS 1.0:an, 3.2 taas TLS 1.1:een ja niin edespäin.

Näiden lukujen jälkeen seuraa Client Random, eli 32 bittiä satunnaisia arvoja: *00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f*. Tätä sarjaa käytetään myöhemmin symmetrisen avaimen luonnissa.

Satunnaisarvoja seuraa istunnon tunniste, Session ID. Esimerkissä tämä on *00* ja tarkoittaa tunnisteen pituutta. Pituus voisi olla myös *FF* viitaten 255 tavun mittaiseen tunnisteeseen.

Istunnon tunnisteen jälkeen seuraa lista tuetuista protokollista avainten vaihtoon, avaimen salaukseen sekä viestin todentamiseen. Sarja *00 20 cc a8 cc a9 c0 2f c0 30 c0 2b c0 2c c0 13 c0 09 c0 14 c0 0a 00 9c 00 9d 00 2f 00 35 c0 12 00 0a* sisältää tiedon listan pituudesta (*00 20 eli 32 tavua*) ja itse listan tuetuista kryptografisista metodeista yhteyden luomiselle:

- 00 20 - 0x20 (32) tavua pitkä lista kryptografisista metodeista
- cc a8 - TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- cc a9 - TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
- c0 2f - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

- c0 30 - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- c0 2b - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- c0 2c - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- c0 13 - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- c0 09 - TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- c0 14 - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- c0 0a - TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- 00 9c - TLS_RSA_WITH_AES_128_GCM_SHA256
- 00 9d - TLS_RSA_WITH_AES_256_GCM_SHA384
- 00 2f - TLS_RSA_WITH_AES_128_CBC_SHA
- 00 35 - TLS_RSA_WITH_AES_256_CBC_SHA
- c0 12 - TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- 00 0a - TLS_RSA_WITH_3DES_EDE_CBC_SHA.

Listan jälkeen seuraavat kaksi heksadesimaalia ilmoittavat pakkausmetodeista. Luvut *01 00* kertovat pakkausmetodien listan pituuden (*01*) sekä pakkausmetodit (*00*). Heksadesimaali *00* ilmoittaa, että pakkausta ei enää käytetä, koska pakkauksella on piirteitä, jotka heikentävät salausta. Pakattua dataa oli mahdollista hyödyntää ja kaapata istuntoja CRIME-hyökkäyksellä [89, 90].

Pakkausmetodien jälkeen ilmoitetaan laajennuksista kahdella heksadesimaaliluvulla *00 58*. Tämä ilmoittaa laajennusten olevan 88 tavua dataa. Esimerkin laajennuksista löytyy tieto palvelimen nimestä (*Server Name*), pyyntö palvelimen sertifikaattia koskevista OSCP-tiedoista (*Online Certificate Status Protocol*), tieto tuetuista elliptisen käyrän ryhmistä, määrittely elliptisen käyrän pisteistä ja sen tiedon siirtämisestä, tieto tuetuista allekirjoitusalgoritmeista, istunnon uudelleenneuvotteluun käytettävistä tiedoista sekä pyyntö allekirjoitetusta sertifikaatin aikaleimasta [47].

Vastauksena Client Hello -viestiin palvelin lähettää oman Server Hello -viestinsä takaisin. Vastaus sisältää tiedon valitusta protokollaversiosta, palvelimen omasta 32-bittisestä satunnaisdatasta, istunnon tunnisteesta, valitusta salaussarjasta, valituista pakkausmetodeista sekä listan laajennuksista.

Palvelin lähettää Server Hello -viestinsä jälkeen sertifikaattinsa. Sertifikaattiviesti sisältää tiedot rekisteriotsakkeesta (Record Header), kättelyotsakkeesta (Handshake Header), koko sertifikaattiotsakkeen pituudesta, seuraavan sertifikaatin pituudesta sekä itse sertifikaatin.

Sertifikaatin jälkeen palvelin luo yksityisen ja julkisen avainparin, jotta osapuolet voivat vaihtaa tietoja yhteisesti valitusta numerosta ilman, että ulkopuoliset saavat sitä tietoonsa. Yksityinen ja julkinen avainpari luodaan käyttäen elliptisen käyrän algoritmia x25519.

Avainparin luomisen jälkeen palvelin lähettää sovellukselle tarvittavat tiedot symmetrisen avaimen luomista varten. Symmetrinen avainpari generoidaan käyttäen kummankin osapuolen yksityistä avainta ja vastapuolen julkista avainta. Lähetetty viesti sisältää tiedot rekisteriotsakkeesta, kättelyotsakkeesta sekä elliptisen käyrän yksityiskohdista Viesti sisältää myös palvelimen julkisen avaimen sekä allekirjoituksen, joka varmistaa palvelimen olevan julkisen avaimen omistaja. Lopuksi palvelin lähettää viestin Server Hello Done, joka ilmoittaa palvelimen olevan valmis jatkamaan kättelyä.

Sovellus vuorostaan luo oman yksityisen ja julkisen avainparinsa. Yksityinen avain luodaan valitsemalla satunnainen kokonaisluku väliltä 0 – (2^{256}-1). Luku valitaan generoimalla 32 tavua eli 256 bittiä satunnaisdataa. Julkinen avain valitaan kertomalla piste $x = 9$ elliptiseltä käyrältä x25519 käyttäen yksityistä avainta.

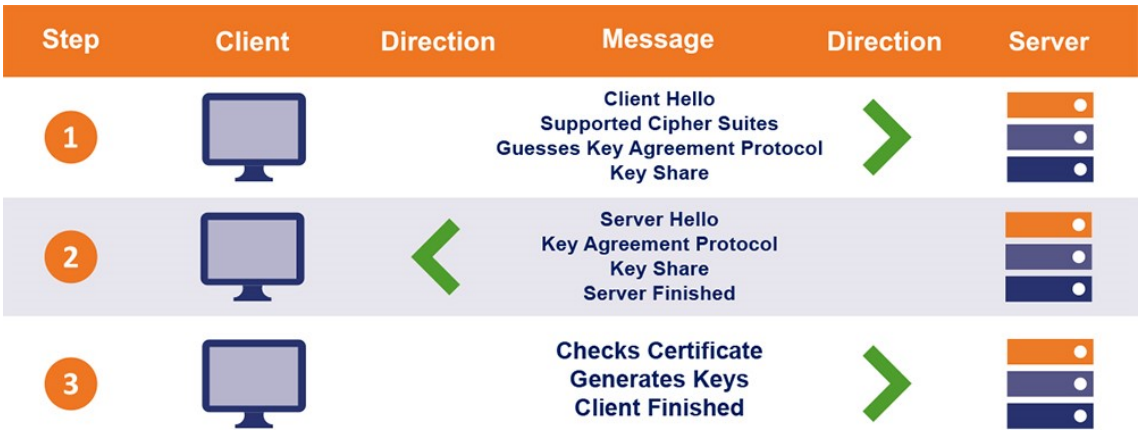
Sovellus lähettää oman julkisen avaimensa palvelimelle. Jaettu salausavain generoidaan käyttämällä osapuolen omaa yksityistä avainta ja vastapuolen julkista avainta. Sovelluksella on nyt tiedossa tarvittavat osat jaetun avaimen luomiseen. Tämän luomisprosessin jälkeen sovellus ilmoittaa palvelimelle olevansa valmis käyttäen ChangeCipherSpec-viestiä. Lopuksi sovellus lähettää Client Handshake Finished -viestin ilmoittaakseen olevansa valmis.

Palvelin laskee symmetrisen avaimen saamiensa tietojen avulla. Tämän jälkeen palvelin lähettää sovellukselle viestit salauksen vaihtamisesta sekä ilmoittaa olevansa valmis yhteyden käyttämiseen. Tästä eteenpäin osapuolet voivat lähettää salattua dataa keskenään, kunnes sovellus ilmoittaa sulkevansa yhteyden.

Koko prosessi vei kaksi RTT-ajanjaksoa. Tämä aika vaihtelee riippuen fyysisistä etäisyyksistä sekä välissä olevan verkon reitityksestä. Se on usein alle sekunnin luokkaa [47].

3.3.2 TLS 1.3 -version kättely

Uusimman version kättely on saatu tiivistettyä yhteen RTT:hen aiemmasta kahdesta. Yhteyden luomiseen tarvittu aika saatiin pudotettua puoleen siirtämällä tarvittuja tietoja kerralla enemmän osapuolten välillä. Kuva 8 esittää uuden kättelyprosessin.



Kuva 8. *TLS 1.3 -kättelyprosessi [88].*

Ennen kättelyn aloittamista sovellus generoi tarvittavan yksityisen ja julkisen avainparinsa avainvaihtoa varten käyttäen samoja metodeja kuin esimerkiksi TLS 1.2 -kättelyssä.

Kuten aikaisempien versioiden kättelyissä, sovellus aloittaa lähettämällä Client Hello -viestin. Tämän viestin mukana lähetetään myös lista tuetuista salausalgoritmeista ja arvauksista, mitä avaintenvaihtoprotokollaa palvelin todennäköisesti tukee. Sovellus lähettää samalla myös oman osansa symmetrisen avaimen luomiseen tarvittavista palasista sekä listan tuetuista protokollaversioista.

Viestin saatuaan palvelin generoi yksityisen ja julkisen avaimen. Vastatessaan Client Hello -viestiin omalla Server Hello -viestillään palvelin lähettää myös tiedon valitsemastaan protokollasta symmetrisen avaimen muodostukseen. Viestin mukana lähetetään myös palvelimen tiedot symmetrisen avaimen luomista varten, palvelimen sertifikaatti sekä Server Finished -viesti. TLS 1.2 -kättelyssä Server Finished -viesti oli vasta kuudennessa askeleessa kättelyä. Nyt se on saatu siirrettyä toiseen askeleeseen, ja samalla on säästetty neljä askelta ja yksi kokonainen RTT.

Sovellus tarkistaa palvelimen sertifikaatin, generoi avaimen tarjottujen tietojen perusteella ja lähettää Client Finished -viestin. Tästä eteenpäin salatun datan siirtäminen voi alkaa [88].

TLS 1.3 -kättelyllä aikaa säästyy useita satoja millisekunteja. Ihmismielelle tämä ei vaikuta paljolta, mutta vuonna 2006 Marissa Mayer, Googlen sen aikainen varapääjohtaja, paljasti yhtiön huomanneen puolen sekunnin viiveen laskevan sivuston liikennettä jopa 20% [91, 92]. Mitättömältä tuntuva aika riitti käännyttämään joka viidennen sivuston käyttäjän.

3.4 TLS-optimoinnit

TLS-protokollaan liittyvät optimoinnit voidaan jakaa kahteen osaan: yhteyden luomiseen ja tiedon salaamiseen. Salattua yhteyttä luotaessa osapuolten tulee varmistua useista asioista. Osapuolten identiteetti varmistetaan sertifikaateilla, tiedon salaaminen suoritetaan yhteisesti valituilla algoritmeilla ja symmetrinen salausavain muodostetaan molemman osapuolen avulla käyttäen ensiksi asymmetrisiä salausavaimia. Kättelyprosessi on suurin osatekijä TLS-yhteyden luomiseen kuluvassa ajassa, ja tätä prosessia voidaan nopeuttaa eri tavoin [93].

käyttäjän.

3.4.1 Yhteyden luonti

Yhteyden luomiseen liittyvä ensimmäinen suuri optimointi on False Start, joka mahdollistaa tiedon lähettämisen jo ennen kuin kättely on suoritettu loppuun. Mikäli toinen osapuoli odottaa vain Finished-tietoa vastapuolelta ja on suorittanut oman osuutensa kättelystä, voidaan dataa lähettää jo ennakkoon. Datan lähetys voidaan aloittaa heti, kun ChangeCipherSpec- sekä Finished-viestit on lähetetty vastapuolelle. Tämä metodi poistaa kättelyyn kuluneesta ajasta yhden kokonaisen RTT:n, sillä lähettäjän ei tarvitse odottaa vuoroaan, vaan voi sijoittaa lähetetyn tiedon vastaanottajan jonoon. Parhaimmillaan tämä metodi pudottaa kättelyn pituuden yhteen RTT:hen.

Istunnon jatkaminen (Session Resumption) on toinen merkittävä optimointi. Jos sovellus on aiemmin ollut yhteydessä palvelimeen, voidaan käyttää lyhennettyä kättelyä, jossa Client-ID -tiedon perusteella käytetään aiemman yhteyden parametreja. Näin kättely kestää vain yhden RTT:n mittaisen ajan ja molemmat osapuolet säästävät prosessointiajan, joka kuluisi salausavaimien luomisessa.

Näillä yhteyden optimointitavoilla voidaan tarjota jatkuvasti 1-RTT -kättelyitä uusille ja vanhoille kävijöille sekä istunnon uusimiseen perustuvaa säästöä laskentatehossa ja ajassa, mikäli vanhoja istuntoparametreja voidaan hyödyntää [94].

TLS 1.3 toi mukanaan myös uuden 0-RTT -ominaisuuden. Tämä ominaisuus mahdollistaa aiemman yhteyden käytön jatkamisen, mikäli sovellus ja palvelin yhä jakavat saman PSK:n (Pre-Shared Key). Tämä pudottaa yhteyden luomiseen käytetyn ajan lähes nollaan. TLS 1.3 mahdollistaa sovellusdatan lähettämisen ensimmäisen paketin mukana ja vastaanottamisen palautuvan yhteydenluontiviestin yhteydessä.

Tämä ominaisuus on kuitenkin saanut osakseen paljon kritiikkiä, sillä jatkuva salassapito ei ole mahdollista eikä mikään vahvasti estä toistohyökkäyksien toteuttamista [95, 2].

3.4.2 Datan pakkaaminen ja algoritmien nopeus

Nopein bitti on se, jota ei koskaan tarvinnut lähettääkään [96]. Aiemmat versiot salausprotokollista tukivat tiedon pakkaamista, jotta lähetettävää olisi ollut vähemmän. Tästä syystä TLS 1.2 -kättely sisältää yhä kentän pakkausmetodeja varten, vaikka on jo yleisesti hyväksytty pakkaamisen olevan turvaton toimenpide. CRIME-hyökkäyksillä oli mahdollista purkaa paketeista tietoa esimerkiksi istuntoevästeistä, vaikka hyökkääjä ei olisikaan tiennyt mitään salaukseen käytetyistä tiedoista [89]. Myös algoritmien nopeuksilla, lohkokoolla ja salausavainten koolla on eroja [liitteet A ja B]. 256-bittisen salausavaimen käyttö 128-bittisen sijaan kasvattaa prosessointiaikoja. Toisaalta 64-bittisellä prosessorilla 512-bittinen saattaa myös olla nopeampi [97].

3.4.3 Tiivistefunktiot

Tiivistefunktioita vertaillessa tulee käyttää samaa laitetta jokaista algoritmia testatessa. Testausprosessissa algoritmin testausaika on vakio ja saatavat vertailuluvut ovat luotujen tiivisteiden kokonaismäärä per algoritmi. Taulukossa 5 on kuvattu eri tiivistefunktioiden generointinopeutta lukumääräisesti kolmen sekunnin ajalta.

Taulukko 5. *md4, md5 sekä hmac(md5) tiivistefunktioiden vertailu [98].*

Testi	Määrä
Doing md4 for 3s on 16 size blocks:	11817261 md4's in 2.99s
Doing md4 for 3s on 64 size blocks:	9164094 md4's in 2.99s
Doing md4 for 3s on 256 size blocks:	5436759 md4's in 2.99s
Doing md4 for 3s on 1024 size blocks:	2053975 md4's in 2.99s
Doing md4 for 3s on 8192 size blocks:	304846 md4's in 2.98s
Doing md5 for 3s on 16 size blocks:	8763024 md5's in 3.00s
Doing md5 for 3s on 64 size blocks:	6537570 md5's in 2.99s
Doing md5 for 3s on 256 size blocks:	3710551 md5's in 2.99s
Doing md5 for 3s on 1024 size blocks:	1334743 md5's in 3.00s
Doing md5 for 3s on 8192 size blocks:	192998 md5's in 2.99s
Doing hmac(md5) for 3s on 16 size blocks:	7157191 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 64 size blocks:	5738849 hmac(md5)'s in 2.99s
Doing hmac(md5) for 3s on 256 size blocks:	3437022 hmac(md5)'s in 2.99s
Doing hmac(md5) for 3s on 1024 size blocks:	1312290 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 8192 size blocks:	193418 hmac(md5)'s in 2.99s

Lohkokokojen kasvaessa generoitujen tiivisteiden lukumäärä laskee. Taulukon 6 vertailu paljastaa *hmac(md5)* -funktion olevan hitain lähes joka tilanteessa.

Taulukko 6. *Tiivistefunktioiden keskinäinen vertailu.*

Testi (3s)	md4 vs md5	md5 vs hmac(md5)	md4 vs hmac(md5)
md4, 16 size blocks:	1,348536875		
md4, 64 size blocks:	1,401758452		
md4, 256 size blocks:	1,465216082		
md4, 1024 size blocks:	1,538854296		
md4, 8192 size blocks:	1,579529322		
md5, 16 size blocks:		1,224366375	
md5, 64 size blocks:		1,139177908	
md5, 256 size blocks:		1,079583139	
md5, 1024 size blocks:		1,017109785	
md5, 8192 size blocks:		0,997828537	
hmac(md5), 16 size blocks:			1,651103205
hmac(md5), 64 size blocks:			1,596852261
hmac(md5), 256 size blocks:			1,581822578
hmac(md5), 1024 size blocks:			1,565183763
hmac(md5), 8192 size blocks:			1,576099432

Ainoastaan *md5* ja *hmac(md5)* -vertailussa *hmac(md5)* voitti hyvin vähäisissä määrin. Kolmen sekunnin aikana generoidut tiivisteet olivat lähes samat lukumäärällisesti. Vertailuun käytetty laskutoimitus on $Ensimmäinen\ algoritmi : Toinen\ algoritmi$ ja mikäli tulos on alle 1, on $Toinen\ algoritmi$ generoinut enemmän tiivisteitä ja täten ollut nopeampi.

Lukumäärät eivät kuitenkaan ole tarkka vertailumääre, joten vertailu on parempi suorittaa käsitellyn datamäärän mukaan.

Taulukko 7. *Käsitelty data tuhansissa tavuissa per sekunti [98].*

Funktio	16 tavua	64 tabua	256 tavua	1024 tavua	8192 tavua
md2	0.00	0.00	0.00	0.00	0.00
mdc2	0.00	0.00	0.00	0.00	0.00
md4	63236.18k	196154.52k	465488.40k	703434.92k	838019.61k
md5	46736.13k	139934.61k	317692.66k	455592.28k	528775.79k
hmac(md5)	38171.69k	122838.24k	294273.46k	447928.32k	529926.51k

Taulukon 7 luvuista nähdään kuinka *hmac(md5)* on ollut nopeampi vain 8192-tavuisten lohkojen käsittelyssä ja vain verrattuna *md5*-funktioon.

3.4.4 Allekirjoitus ja varmistus

Allekirjoitus ja varmistus toteutetaan yksityisen ja julkisen avainparin avulla. Avainpari voidaan generoida eri algoritmein, mutta tämä prosessi ei liity tietojen allekirjoitukseen tai varmistamiseen.

Taulukko 8. *RSA-algoritmin nopeus eri bittimäärissä [98].*

Algoritmi	allekirjoitus	varmistus	allekirjoitus/s	varmistus/s
rsa 512 bits	0,0001	0,000007	9976,6	142275,1
rsa 1024 bits	0,000350s	0,000020s	2858,7	49126,4
rsa 2048 bits	0,002291s	0,000072s	436,5	13813,4
rsa 4096 bits	0,017219	0,000273	58,1	3659,7

Taulukosta 8 nähdään, kuinka algoritmissa käytettyjen bittien määrä nostaa operaatioon kuluvaa aikaa merkittävästi. 512-bittinen allekirjoitus on 172-kertaisesti nopeampi kuin 4096-bittinen ja varmistus on 39-kertaisesti nopeampaa.

3.4.5 Avaimenvaihto

Avaimenvaihtoon käytettävistä algoritmeista suositaan nykyään elliptisen käyrän metodia. NIST-käyrät (National Institute of Standards and Technology) olivat suuresti käytössä ennen Curve25519:ää [99], mutta NSA:n vakoilun noustessa otsikoihin [100], alkoivat monet tietoturva-alan asiantuntijat epäilemään käyrien ominaisuuksien tarjoavan NSA:lle mahdollisuuden vakoilla verkkoliikennettä huomaamatta [101]. NIST hyväksyi Curve25519:n yleiseksi standardiksi vuonna 2017 [99, 102]. Taulukko 9 esittää NIST:n hyväksymien käyrien nopeusvertailua.

Taulukko 9. *Elliptisten käyrien vertailu* [98].

Elliptinen kurvi	Operaatio	Operaatiota/s
160 bit ecdh (secp160r1)	0,0003s	3239,5
192 bit ecdh (nistp192)	0,0004s	2825,5
224 bit ecdh (nistp224)	0,0002s	6557,5
256 bit ecdh (nistp256)	0,0003s	3225,8

TLS 1.2 ja 1.3 -versioiden käyttämää elliptistä käyrää Curve25519 [104] vertaillessa muihin elliptisiin kurveihin huomataan, kuinka kevyt algoritmi kyseessä on.

Taulukko 10. *Nykyaikaisten elliptisten käyrien vertailu [103].*

Curve	Clock cycles	Size	RAM usage
NIST K-233	≈5,382,144	≈38,600 bytes[b]	≈3700 bytes
NIST B-233	≈13,934,592	≈34,600 bytes[b]	≈2,200 bytes
NIST P-224	≈17,520,000	4812 bytes	422 bytes
256-bit Montgomery	≈21,078,200	14,700 bytes[b]	556 bytes
NIST P-256	≈34,930,000	16,112 bytes	590 bytes
Curve25519	22,791,579	n/a[a]	677 bytes
Curve25519	14,146,844	9,912 bytes	510 bytes
Curve25519	13,900,397	17,710 bytes	494 bytes

Taulukon 10 vertailut ovat toteutettu AVR ATmega -mikrokontrollerilla. Curve25519 on ollut merkittävästi nopeampi sekä käyttänyt vähemmän keskusmuistia (RAM) kuin vertailukohteet. Tämän vuoksi yhtiöt kuten Facebook sekä Google ovat vaihtaneet käyttämään Curve25519-käyrää erinäisissä palveluissaan [105, 106].

4 PROTOKOLLIEN MITTAAMINEN

Työ keskittyy tutkimaan kättelyyn liittyviä eroavaisuuksia TLS 1.1, 1.2 sekä 1.3 -versioiden välillä. Tarkastelussa käytettiin Webpagetest-ohjelmistoa [107], joka mittaa verkkosivun kokonaislatausaikoja sekä yksittäisten tiedostojen latausaikoja. Näiden lisäksi talteen otetaan tiedostokoot, latausjärjestykset, ulkoiset resurssit sekä tarkka verkkodata koko prosessista aina GET-pyyntöjä myöden. Työn mittaukset suoritettiin suljetussa laboratorioympäristössä, jonka avulla vältettiin ylimääräinen verkkodata ja prosessointitehon hukkakäyttö. Näin mittaukset saatiin pidettyä vertailukelpoisina.

4.1 Mitattavat suureet

Testauksessa mitattiin verkkosivun esittämiseen kulunut kokonaisaika sekä yksittäisten komponenttien pyytämiseen, vastaanottamiseen sekä prosessointiin kulunut aika, joka laskettiin teoriaan perustuen. Mitattaviin aikoihin vaikuttaa muun muassa vaadittujen edestakaisten pakettien määrä sekä mittausympäristössä säädettävän RTT:n (Round-trip-time) suuruus. Työssä kiinnostavimmat suureet olivat kättelyyn kulunut aika, prosessointiaika sekä ensimmäisen Request-pyynnön käsittelyyn kulunut kokonaisaika.

Kättely on salatun yhteyden tärkein osa. Tällä toiminnolla sovitaan yhteisistä salausalgoritmeistä, käytettävistä sertifikaateista ja muista arvoista. Kättely tapahtuu eri tavalla TLS 1.1/1.2 ja 1.3 välillä, joka vaikuttaa suoraan kättelyn kokonaiskestoon.

Kokonaiskestoon vaikuttaa asetettu RTT, matkan fyysinen pituus, sivuston koko, lähetys- ja vastaanottoaika sekä prosessointiaika sekä sovellus- että palvelinpäästä. Staattisia muuttujia näistä ovat matkan fyysinen pituus, sisältäen kuparijohdotuksen laitteiden välillä, sekä sivuston koko (88.2 KB).

RTT, oli muutettavissa haluttuihin arvoihin (20, 100, 200, 300 ja 400 millisekuntia [ms]). Mitattavia muuttujia siis ovat lähetys- ja vastaanottoajat sekä kättely- ja prosessointiaika.

Odotus- ja vastaanottoaika vaihtelevat monista syvällä piilevistä syistä. Odotusaika alkaa, kun sovellus on lähettänyt palvelimelle pyynnön resursseista. Se loppuu, kun palvelin vastaa pyyntöön ensimmäisen kerran. Tähän aikaan sisältyy muun muassa säädetty RTT sekä palvelimen prosessointiaika tarvittavien resurssien valmisteluun. Vastaanottoaika alkaa, kun sovellus vastaanottaa ensimmäisen paketin resursseista ja loppuu, kun viimeinen tarvittu resurssipaketti on saapunut. Tässä prosessissa mitataan puhtaasti sovelluksen kykyä käsitellä saapuvat paketit.

4.2 Mittausympäristö

Mittausympäristö sisälsi kolme fyysistä laitetta, joista kahdella ajettiin Xencenter-virtuaalialustaa käyttöjärjestelmien virtualisointiin. Kolmas laite (Lubuntu Router, reititin) toimi RTT:n säätäjänä. Näin saatiin simuloitua eri mittaisia verkkoyhteyksiä, sillä laboratorio-oloissa viiveet olivat alle 2 millisekunnin luokkaa ilman itse luotua viivettä.

TLS 1.3 -protokollan vuoksi mittausympäristöstä päivitettiin OpenSSL sekä Apache2. Uusin versio OpenSSL:stä ei enää tue SSL 3 -protokollaa, joten kaikki mittaukset SSL 3:sta tehtiin eri ympäristössä kuin TLS 1.2 ja TLS 1.3. Nämä tulokset sisällytettiin viittausarvoiksi eikä niitä käytetä vertailuun muuten kuin HTTP/1.1 ja HTTP/2.0 välisten erojen havainnoimiseen.

Apache2-konfiguraatiosta asetettiin “SSLSessionCacheTimeout” oletusarvoisesta 300 sekunnista 15 sekuntiin, sillä nopeasti toistetut mittaukset olisivat muuten saattaneet käyttää aiempaa yhteyttä [108].

Työn sovellukset pyrittiin pitämään ajan tasalla ja lukumäärältään vähäisinä. Ainoat tarvitut sovellukset olivat OpenSSL 1.1.1, Apache2 2.4.38, WebPageTest 17.12, iproute v1.7.1 sekä Google Chrome 72.0.3626.121.

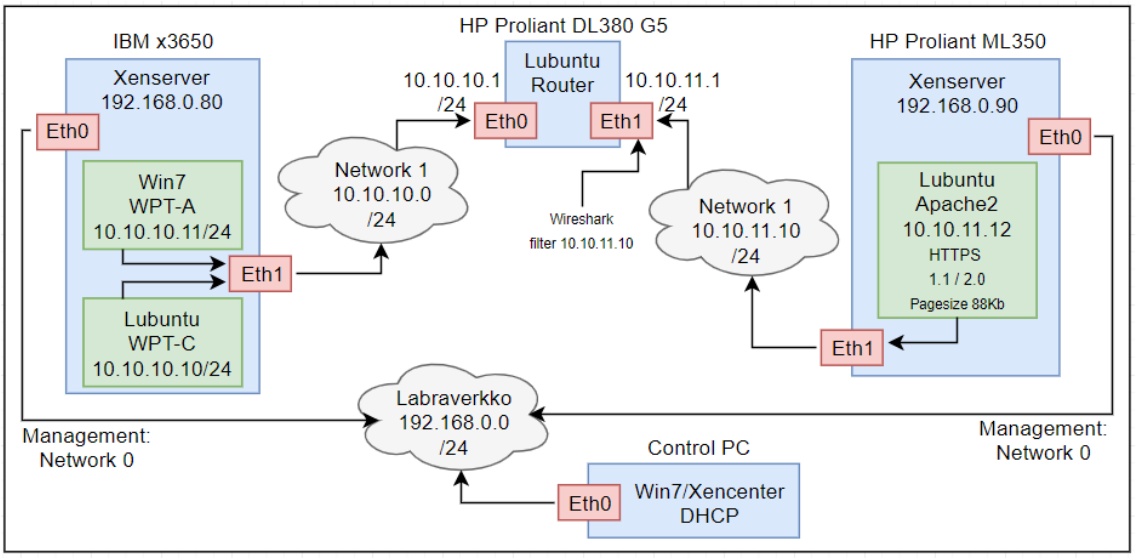
OpenSSL oli työn suurin tekijä, sillä SSL/TLS-konfiguraatio oli täysin OpenSSL:n salaussarjojen varassa. Apache2 toimi verkkosivupalvelimena, johon otettavia yhteyksiä mitattiin. Lisäksi sen konfiguraatiosta säädettiin HTTP/1.1 sekä HTTP/2.0 tukea. WebPageTest toimi mittaustyökaluna, joka hallitsi Chrome-selainta. Iproute-ohjelman ominaisuuksien avulla saatiin säädettyä RTT-aikoja. Työssä käytetyt laitteet ja niiden suorituskykyyn vaikuttavat asiat on esitelty Taulukossa 11.

Taulukko 11. Mittausympäristön palvelinlaitteiden resurssit

IBM x3650	Prosessori: Intel Xeon E5420 @2.50 GHz RAM: 10 Gb Tallennusmedia: 1821 Gb SSD, raid 0
HP Proliant DL380 G5	Prosessori: Intel Xeon E5420 @2.50 GHz RAM: 10 Gb Tallennusmedia: 500 Gb SSD, raid 0
HP Proliant ML350	Prosessori: Intel Xeon 5130 @2.00 GHz RAM: 8 Gb Tallennusmedia: 500 Gb SSD, raid 0

Mittausympäristö on esitetty Kuvassa 9. Xenserver-laitteet olivat hallinnassa jaetun verkon kautta (Management: Network-0). Ainoastaan Lubuntu Router -reititin on eroteltu Control PC:n näkymästä, johon kuitenkin pääsee SSH:n kautta käyttämällä esimerkiksi

Win7 PC:tä hyppykoneena. Lubuntu Router -reitittimeen asennettiin Wireshark pakettien seuraamista varten, mutta tätä ominaisuutta ei kuitenkaan käytetty muuten kuin yhteyksien ongelmien ratkomiseen.



***Kuva 9**. Laboratorioverkon esittely.*

4.3 Mittausmetodit

Mittaamisen metodit olivat osittain kvalitatiivisia ja osittain kvantitatiivisia. Mittausympäristö pyrittiin pitämään mahdollisimman muuttumattomana dedikoimalla fyysiset laitteet vain tätä työtä varten. Jokainen mittaus toistettiin viisi kertaa. Näin saatiin puhtaita tuloksia, joista otettiin keskiarvoja. Muuttumattomina arvoina pidettiin ladattavan verkkosivun kokonaiskokoa, fyysistä etäisyyttä laitteiden välillä sekä työkuormaa jokaisella laitteella. Muutettavina arvoina pidettiin Lubuntu Router -reitittimen kautta määriteltyä RTT:n aikaa sekä protokollia. Muuttuvina arvoina seurattiin verkkosivuresurssien latausaikoja eri protokollayhdistelmien kohdalla.

Jokainen salausprotokolla mitattiin käyttäen HTTP/1.1:stä ja HTTP/2.0:aa. Protokollia vaihdettiin muokkaamalla *ssl.conf*-tiedostoa, johon lisättiin rivit:

```
SSLProtocol -all +TLSv1.1,
SSLProtocol -all +TLSv1.2 ja
SSLProtocol -all +TLSv1.3.
```

Käyttämällä #-merkkiä kommentoitiin pois rivit, joiden protokollia ei mitattu. Myös istunnon pituus asetettiin 15 sekunnin mittaan rivillä *SSLSessionCacheTimeout 15*. Näin

saatiin estettyä yhteyden uudelleenkäyttö yli 15 sekunnin jälkeen. Konfiguraatiotiedoston viimeiset rivit näyttivät siis seuraavilta:

```
SSLSessionCacheTimeout 15
SSLProtocol -all +TLSv1.1
#SSLProtocol -all +TLSv1.2
#SSLProtocol -all +TLSv1.3.
```

HTTP-protokollia vaihdettiin muokkaamalla *httpd.conf*-tiedostosta *Protocols*-riviä. SSL/TLS-istunnon uudelleenkäyttötapa määriteltiin riveillä *SSLSessionCache* ja *SSLSessionTickets*. Konfiguraation lopussa olivat alla olevat rivit,

```
Protocols h2
#Protocols http/1.1
#SSLSessionCache none
#SSLSessionTickets off
```

5 TULOKSET JA VERTAILU

Mittauksista saatiin kolme eri tiedostoa jokaiselle kategorialle. HAR (HTTP Archive) -tiedosto sisälsi koko mittauksen tiedot tekstimuodossa. Details.csv sisälsi saman datan kuin HAR, mutta eri muodossa. Tärkeimmät tiedot jokaisen mittauskerran kohdalta tallennettiin Summary.csv -tiedostoon.

Mittaustuloksista ne, joiden RTT-arvo oli joko *100ms* tai *300ms,* toimivat lähinnä lisävarmistuksena *20ms*, *200ms* ja *400ms* mittauksille. Lopulliset vertailut suoritettiin *20ms*, *200ms* ja *400ms* tulosten avulla.

Tulosten tarkasteluun käytettiin useampaa HAR Viewer -sovellusta, joista kaikki olivat ilmaisia:

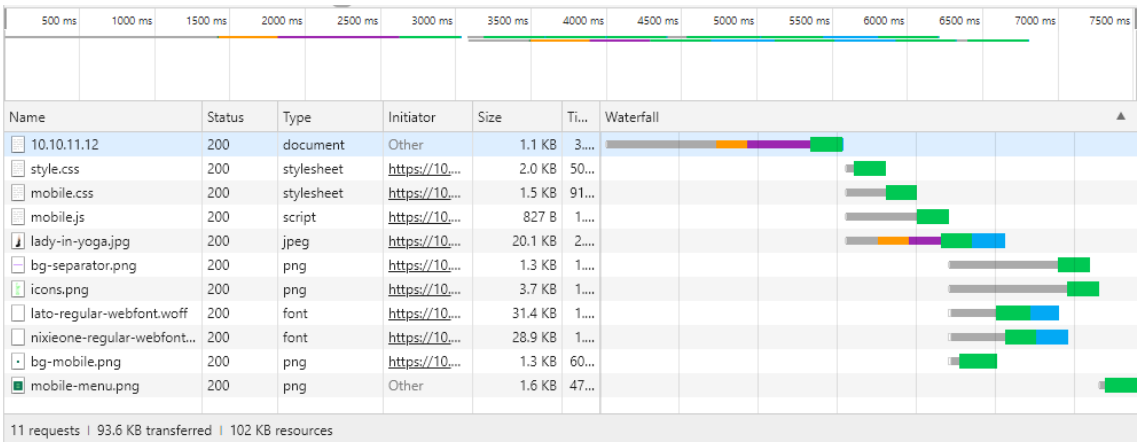
- G Suite Toolbox HAR Analyzer [109]
- Sitespeed.io Compare [110]
- HTTP Archive Viewer 2.0.17 [111]
- Google Chrome Developer Tools [112].

Mittauksista kerättiin tärkeimmät tiedot yhteen Excel-tiedostoon vertailua varten.

Esitetyt tulokset ovat suuntaa antavia, sillä testien toistomäärä on vain 5. Näistä mittauksista on otettu keskiarvot vertailun helpottamista varten, mutta vähäisen toiston vuoksi tulokset saattavat poiketa tosimaailman tuloksista. Suuret poikkeamat yksittäisen mittauksen tuloksissa on jätetty huomiotta, jotta tulokset eivät vääristy.

5.1 HTTP/1.1

HTTP/1.1-mittauksen tuloksissa oli odotettavissa pitkiä kokonaisaikoja koko sivun lataamiselle, sillä se ei tue kanavointia, otsakkeiden tiivistämistä, tärkeysjärjestystä eikä pakettihallintaa [113]. Tämä ei kuitenkaan vaikuttanut yhteyden luomiseen, sillä ensimmäinen GET-paketti käsitellään kuten HTTP/2.0:ssakin.



Kuva 10. *Vesiputouskuva verkkosivun lataamisesta käyttäen HTTP/1.1 ja TLS 1.1 – protokollaa sekä 200ms RTT:tä. Harmaa väri on jonotusaikaa, keltainen on TCP-kättelyyn kulunut aika, violetti on TLS-kättelyyn käytetty aika, vihreä on odotusaika pyynnöstä vastaukseen ja sininen on resurssin lataamiseen käytetty aika.*

Kuvasta 10 nähdään kuinka vanhimmilla protokollilla ja 200 millisekunnin RTT:llä mitattuna koko sivun lataamiseen meni noin 6800ms, eli 6,8 sekuntia. Request-pyyntö on kokonaisuudessaan kestänyt 3030 millisekuntia. Jonotustilassa on kulunut 1410 millisekuntia. Tämän jälkeen yhteyden luomiseen on kulunut 1210 millisekuntia, josta TLS-kättely on vienyt 809 millisekuntia. Vastaanottamiseen on kulunut 403 millisekuntia. Ilman jonotusaikoja ensimmäisen Request-pyynnön käsittelyyn on siis kulunut 1620 millisekuntia:

$$Pyynnön\ käsittely = Kokonaisaika - Jonotusaika$$

$$3030ms - 1410ms\ =\ 1620ms.$$

Yhteyden luomiseen on käytetty 1217 millisekuntia:

$$Yhteyden\ luominen = Pyynnön\ käsittely - Vastaanottaminen$$

$$1620ms - 403ms = 1217ms.$$

Yhteyden luomisesta 809 millisekuntia oli kulunut SSL/TLS-kättelyyn, joten TCP-yhteyden luomiseen oli käytetty 408 millisekuntia:

$$TCP\text{:}n\ luonti = Yhteyden\ luominen - Kättely$$

$$1217ms - 809ms =\ 408ms.$$

Vesiputouskuvasta nähdään, että lähes jokainen resurssi on ladattu eri aikaan ja erillisellä Request-pyynnöllä. Jokainen resurssi on päätynyt jonoon, sillä HTTP/1.1:n

vuoksi useita resursseja ei ole voitu ladata samanaikaisesti. Tämä käy hyvin ilmi ensimmäisen neljän resurssin lataamisen yhteydessä (*style.css, mobile.css, mobile.js* ja *lady-in-yoga.jpg*).

Jokainen resurssi on jonottanut, kunnes aiempi on saatu ladattua. *Lady-in-yoga.jpg* -kuvan lataamisen yhteydessä on jopa solmittu uusi yhteys. Tämä näkyy kuvassa violetilla värillä, joka ilmaisee TLS-kättelyä. Jonotus käy ilmi harmaasta väristä, joka tarkoittaa “Stalled / Queued”, eli resurssi on pyydetty, muttei vielä lähetetty. Keltainen väri taas on TCP-yhteyden solmimista, joka on yhden RTT:n verran selaimen näkökulmasta.

Ensimmäisen jonotusajan poistamalla saamme koko sivun lataamiseen kuluneen ajan:

$$Latausaika = Kokonaisaika - Ensimmäisen\ pyynnön\ jonotusaika$$

$$6800ms - 1410ms = 5390ms.$$

5.1.1 TLS-protokollat HTTP/1.1:n alaisuudessa

TLS-protokollia vertaillessa mittaustulokset kävivät oudoiksi. Alun perin TLS 1.2 -mittausten kättelyn osuus oli vain 50% odotetusta arvosta, kunnes huomattiin ottaa SSLSessionCache-ominaisuus pois käytöstä. Kyseinen ominaisuus tallentaa Client-ID:n palvelimen välimuistiin. Jatkossa tätä arvoa voidaan käyttää vanhojen yhteyksien uudelleen luomiseen. Tällä tapaa ohitetaan yksi kokonainen RTT-aika. SSLSessionCachen vaihtaminen pois päältä pakotti uuden TLS-kättelyn läpikäymisen jokaisen yhteyden luonnissa.

Taulukko 12. *TLS-protokollien käyttämä aika (ms) eri toimintoihin HTTP/1.1:n yhteydessä eri RTT-aikoja (20, 200, 400ms) käyttäen.*

20ms	Connection	Handshake	Waiting	Receiving	Total
TLS 1.1	70	49	23	18	160
Session Resumption Disabled, TLS 1.2	69	48	23	18	158
Session Resumption Enabled, TLS 1.2	51	27	16	18	112
TLS 1.3	49	27	18	18	112

200ms	Connection	Handshake	Waiting	Receiving	Total
TLS 1.1	609	408	203	37	1257
Session Resumption Disabled, TLS 1.2	608	408	203	36	1255
Session Resumption Enabled, TLS 1.2	409	207	204	21	841
TLS 1.3	409	207	192	38	846

400ms	Connection	Handshake	Waiting	Receiving	Total
TLS 1.1	1210	809	403	25	2447
Session Resumption Disabled, TLS 1.2	1209	808	403	25	2445
Session Resumption Enabled, TLS 1.2	814	408	407	21	1650
TLS 1.3	810	408	394	22	1634

Taulukosta 12 nähdään, että TLS 1.1 sekä TLS 1.2 (Session Resumption Disabled) -yhteyksien kättely on kestänyt kahden RTT:n verran. Taulukko esittää myös muuhun prosessointiin kuluneen ajan. Prosessointi sisältää tarvittujen algoritmien käyttöön kuluneen ajan sekä paketin muodostamiseen kuluneen ajan. Nämä tapahtumat ovat kestäneet keskimäärin 8 millisekuntia. Pienin prosessointiin mitattu aika on ollut 7 millisekuntia ja suurin 9 millisekuntia.

Taulukko 13 esittää eri TLS-protokollien käyttämiä algoritmeja. TLS 1.2 ja 1.3 ovat täysin samoja, sillä TLS 1.3 poisti vanhoja salausalgoritmeja käytöstä. Google Chrome -selain pyrkii tukemaan tätä siirtymistä priorisoimalla samoja protokollia TLS 1.2 -yhteyksissä.

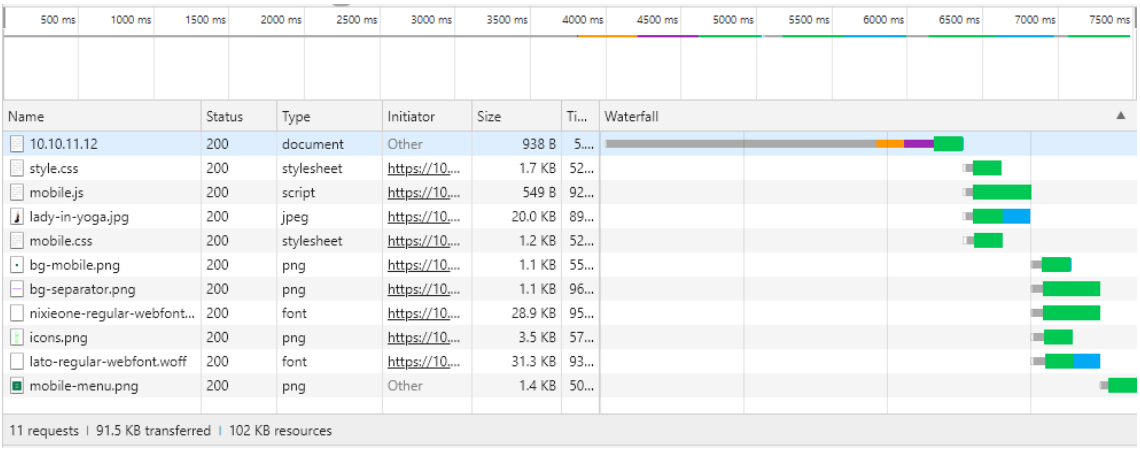
Taulukko 13*. Mittauksessa esiintyneet algoritmit.*

Käytetyt algoritmit			
TLS 1.1	ECDHE_RSA	X25519	AES_128_CBC with HMAC_SHA1
Session Resumption Disabled, TLS 1.2	ECDHE_RSA	X25519	CHACHA20_POLY1305
Session Resumption Enabled, TLS 1.2	ECDHE_RSA	X25519	CHACHA20_POLY1305
TLS 1.3	ECDHE_RSA	X25519	CHACHA20_POLY1305

5.2 HTTP/2.0

HTTP/2.0:n odotettiin vievän vähemmän kokonaisaikaa koko sivun lataamisessa HTTP/1.1:een verrattuna. HTTP/2.0 toi tuen kanavointiin sekä palvelimen resurssien

työntämiseen, jolloin resursseja saadaan ladattua rinnakkain ilman erillistä selaimen lähettämää pyyntöä.



Kuva 12. *Vesiputouskuva verkkosivun lataamisesta käyttäen HTTP/2.0- ja TLS 1.3 – protokollaa sekä 200ms RTT:tä.*

Kuvasta 12 voidaan todeta, että resurssien lataaminen on tapahtunut yhtä aikaa kahdessa keskimmäisessä erässä. Resurssit eivät ole jumiutuneet jonottamaan vuoroaan Request-pyynnön jälkeen, vaan palvelin on työntänyt tarvitut resurssit selaimelle. Lisäksi osa resursseista on ladattu eri järjestyksessä verrattuna HTTP/1.1 + TLS 1.1 -mittauksiin.

Ensimmäisen Request-pyynnön käsittelyyn on tällä kertaa kulunut 1230 millisekuntia:

$$Pyynnön\ käsittely = Kokonaisaika - Jonotusaika$$

$$5030ms - 3800ms\ =\ 1230ms.$$

Yhteyden luomiseen on käytetty 827 millisekuntia:

$$Yhteyden\ luominen = Pyynnön\ käsittely - Vastaanottaminen$$

$$1230ms - 403ms = 827ms.$$

SSL/TLS-kättelyyn kului 407 millisekuntia, joten TCP-yhteyden luomiseen on käytetty 420 millisekuntia:

$$TCP\text{:}n\ luonti = Yhteyden\ luominen - Kättely$$

$$827ms - 407ms =\ 420ms.$$

Koko sivun lataamiseen kuluneen ajan laskeminen tapahtuu kuten aiemminkin:

$$Latausaika = Kokonaisaika - Ensimmäisen\ pyynnön\ jonotusaika$$

$$7373ms - 3800ms = 3573ms.$$

5.2.1 TLS-protokollat HTTP/2.0:n alaisuudessa

Kaikki versiot TLS-protokollasta mitattiin seuraavaksi käyttäen HTTP/2.0:aa. Näkyviä muutoksia kättelyn ja ensimmäisen paketin kokonaisajassa ei odotettu olevan, ja tulokset tukivat tätä oletusta. Muutokset olivat kahden ja yhdeksän millisekunnin välillä. Tulokset on esitetty Taulukossa 14. Työn kannalta kiinnostavat arvot on merkattu värillä.

Taulukko 14. *TLS-protokollien käyttämä aika (ms) eri toimintoihin HTTP/2.0 – yhteydessä eri RTT-aikoja (20, 200, 400ms) käyttäen.*

20ms	Connection	Handshake	Waiting	Receiving	Total
TLS 1.1	70	48	22	18	158
Session Resumption Disabled, TLS 1.2	69	47	22	18	156
Session Resumption Enabled, TLS 1.2	53	27	23	18	121
TLS 1.3	49	28	23	18	118

200ms	Connection	Handshake	Waiting	Receiving	Total
TLS 1.1	610	409	202	27	1248
Session Resumption Disabled, TLS 1.2	610	408	202	27	1247
Session Resumption Enabled, TLS 1.2	409	207	209	24	849
TLS 1.3	409	207	203	25	844

400ms	Connection	Handshake	Waiting	Receiving	Total
TLS 1.1	1210	808	402	24	2444
Session Resumption Disabled, TLS 1.2	1210	807	402	24	2443
Session Resumption Enabled, TLS 1.2	811	408	406	27	1652
TLS 1.3	808	407	403	17	1635

Kuten aiemmin esitetyn HTTP/1.1 mittauksen kohdalla, myös HTTP/2.0 mittauksen tuloksissa on havaittavissa SSLSessionCache-ominaisuus. TLS 1.2:n kättelyn aika on kaksinkertainen RTT-aikaan nähden, kunnes istunnon uudelleenkäyttö laitetaan päälle. Tämä pudottaa kättelyn taas yhden RTT:n mittaiseksi. Kuten HTTP/1.1:n mittauksissa, myös HTTP/2.0:ssa salausalgoritmien prosessointi kestää 7~9 millisekuntia.

Taulukko 15. *Mittauksessa esiintyneet algoritmit.*

Käytetyt algoritmit			
TLS 1.1	ECDHE_RSA	X25519	AES_128_CBC with HMAC_SHA1
Session Resumption Disabled, TLS 1.2	ECDHE_RSA	X25519	CHACHA20_POLY1305
Session Resumption Enabled, TLS 1.2	ECDHE_RSA	X25519	CHACHA20_POLY1305
TLS 1.3	ECDHE_RSA	X25519	CHACHA20_POLY1305

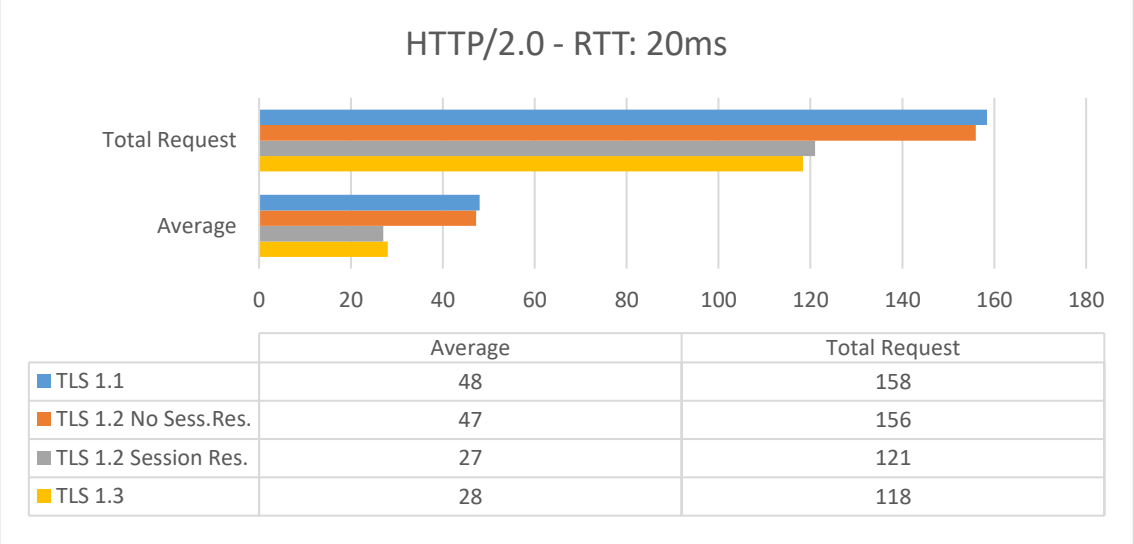
Kuten Taulukko 15 osoittaa, HTTP-protokollan vaihto uudempaan ei muuta kättelyssä käytettyjä algoritmeja. TLS 1.2 ja 1.3 yhä käyttävät Daniel J. Bernsteinin työstämiä algoritmeja x25519 ja ChaCha20 [82].

5.3 Tulosten vertailu

Vertailua varten tulokset järjestettiin RTT-ajan, TLS-protokollan ja HTTP-protokollan perusteella. Jokaisesta mittauksesta otettiin sivuston lataamiseen kulunut kokonaisaika, johon ei ollut laskettu ensimmäisen Request-pyynnön jonotusaikaa mukaan. Latausajat noudattivat pitkälti odotettua kaavaa, mutta joissain mittauksissa datan prosessointi ja lähetysaika vaihtelivat odottamattoman paljon.

5.3.1 TLS-protokollien vertailu

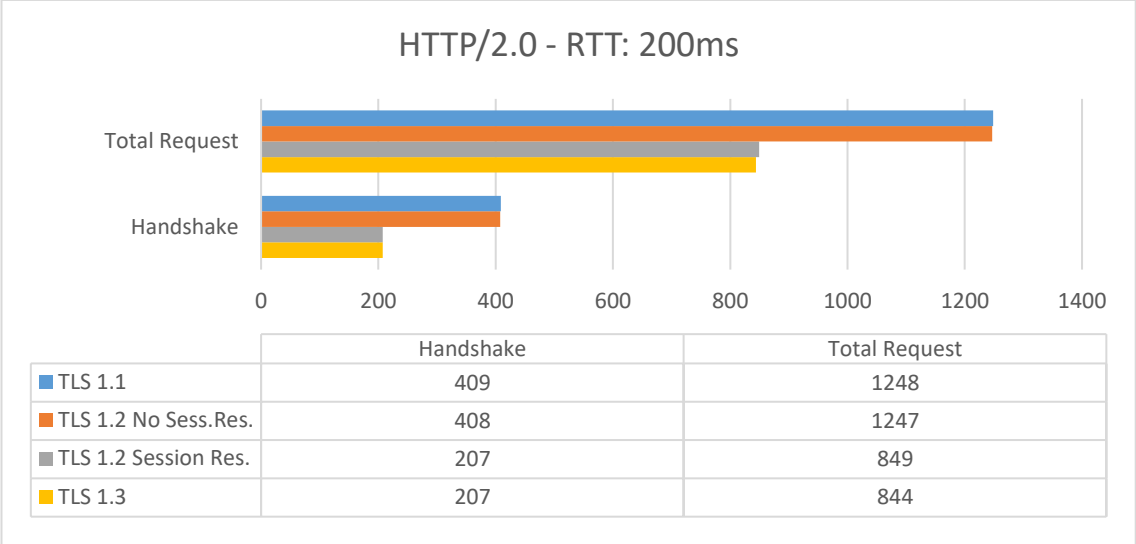
TLS-protokollien vertailua varten tuloksista kerättiin yhteen vain yhteyden luomiseen kulunut kokonaisaika sekä kättelyyn kulunut aika. Tulokset järjestettiin protokollien mukaan kronologisesti, ja jokaisesta RTT-arvosta tehtiin oma taulukkonsa.



	Average	Total Request
TLS 1.1	48	158
TLS 1.2 No Sess.Res.	47	156
TLS 1.2 Session Res.	27	121
TLS 1.3	28	118

Kuva 13. *TLS-protokollat vertailtuna 20ms RTT:llä sekä HTTP/2.0:lla.*

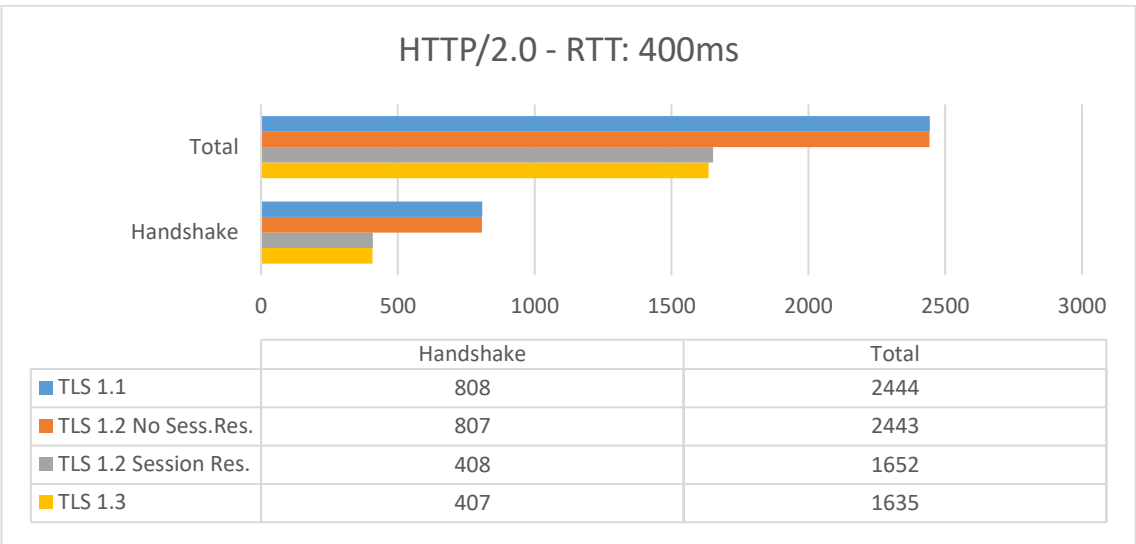
20 millisekunnin RTT-arvolla protokollien ajoissa näyttäisi olevan mittaviakin eroja. Muutaman millisekunnin muutos kuuluu vielä mittavirheen piiriin, mutta yli 40% erot eivät. Kuvasta 13 nähdään, kuinka TLS 1.1 ja 1.2 kättelyjen kestot ovat ilman istunnon uusintaa noin puolet pidempiä kuin TLS 1.3 ja 1.2:ssa, joissa istunnon uusinta on käytössä.



	Handshake	Total Request
TLS 1.1	409	1248
TLS 1.2 No Sess.Res.	408	1247
TLS 1.2 Session Res.	207	849
TLS 1.3	207	844

Kuva 14. *TLS-protokollat vertailtuna 200ms RTT:llä sekä HTTP/2.0:lla.*

Kuvasta 14 nähdään kuinka 200 millisekunnin RTT:n mittauksissa yhdenkin millisekunnin muutos on vielä havaittavissa, mutta tuloksia voidaan pitää yhteneväisinä. TLS 1.3 on käyttäytynyt kuten TLS 1.2 istunnon uudelleenkäytön ollessa päällä.



	Handshake	Total
TLS 1.1	808	2444
TLS 1.2 No Sess.Res.	807	2443
TLS 1.2 Session Res.	408	1652
TLS 1.3	407	1635

Kuva 15. *TLS-protokollat vertailtuna 400ms RTT:llä sekä HTTP/2.0:lla.*

Kuvassa 15 näkyvät 400 millisekunnin mittaukset noudattelevat aiempia tuloksia. TLS 1.1 ja 1.2 ilman istunnon uusimista ovat tasavertaisia kättelyajaltaan. Myös TLS 1.3 ja 1.2 istunnon uusimisella ovat tasavertaisia kättelyajaltaan.

Kuvia 13, 14 ja 15 tarkastellessa huomataan, kuinka RTT-aikojen lisäksi jokaiseen kättelyyn kuluu myös ylimääräiset 7~8 millisekuntia. Tämä aika muodostuu asymmetristen avainten generoinnista, avainten vaihtoon käytetystä ajasta sekä symmetrisen avaimen generoinnista.

5.3.2 Latausajat: HTTP/1.1 verrattuna HTTP/2.0

Taulukosta 16 nähdään, että HTTP-protokollien nopeuserot ovat 32~35% luokkaa. Latausaikojen odotettiin myös lyhenevän TLS-versioiden mukaan, sillä TLS-protokollan kättelyn piti vähentää ainakin yhden RTT:n verran kokonaisajasta.

Taulukko 16. *Sivun lataukseen kuluneen ajan vertailu eri HTTP-versioilla.*

Round-trip-time (ms) / Protokolla	Sivun latausaika (ms)			
20ms	HTTP/1.1	HTTP/2.0	Erotus	%-Erotus
TLS 1.1	1120	728	392	35,00
TLS 1.2 No Session Res.	1115	730	385	34,53
TLS 1.2 Session Res.	1061	695	366	34,50
TLS 1.3	1019	660	359	35,23
200ms	HTTP/1.1	HTTP/2.0	Erotus	%-Erotus
TLS 1.1	3061	2055	1006	32,87
TLS 1.2 No Session Res.	3035	2036	999	32,92
TLS 1.2 Session Res.	3173	2150	1023	32,24
TLS 1.3	3084	2026	1058	34,31
400ms	HTTP/1.1	HTTP/2.0	Erotus	%-Erotus
TLS 1.1	5489	3717	1772	32,28
TLS 1.2 No Session Res.	5556	3764	1792	32,25
TLS 1.2 Session Res.	5113	3458	1655	32,37
TLS 1.3	5200	3424	1776	34,15

Odotuksista huolimatta taulukosta löytyvä TLS 1.2 –protokollan mittaus 200ms RTT:llä ja istunnon uusimisella oli kokonaiskestoltaan noin 100 millisekuntia enemmän kuin muut 200 millisekunnin RTT:llä toteutetut mittaukset. Tämä tulos käy odotuksia vastaan, sillä aikojen odotettiin lyhenevän protokollayhdistelmän moderniuden myötä. Myös 400ms RTT:n mittaus HTTP/1.1- ja TLS 1.3 –protokollilla on vähintään 87ms suurempi kuin odotettu. Nämä tulokset ovat kuitenkin vain muutaman prosentin poikkeavia odotetuista tuloksista, joten kyseessä on todennäköisesti mittavirhe.

6 JOHTOPÄÄTÖKSET

Yhä useammin toistuvat tietoturvarikkeet tulevat jatkamaan uutissivustojen pääotsikkoina. Tämä on johtunut eritasoisten haavoittuvuuksien löytymisestä erinäisistä protokollista sekä ohjelmistoista. Myös inhimilliset virheet ovat olleet usein syynä otsikoihin, väärin konfiguroidut ja päivittämättä jätetyt palvelimet yliedustettuina. Täydellinen esimerkki katastrofitason tietoturvaskandaalista on vuoden 2017 Equifax-tapaus, jossa palvelimet olivat jääneet päivittämättä ja järjestelmän kirjautumistiedot olivat ensiasennuksen jälkeen jätetty oletusarvoonsa [114, 115, 116, 117].

TLS-protokollan uusin versio pyrkii omalta osaltaan vähentämään väärin konfiguroituja palvelimia ja nopeuttamaan verkon toimintaa. Yhteyden salaamiseen käytetyt algoritmit ja protokollat ovat vastustuskykyisempiä väärälle konfiguraatiolle ja salatun yhteyden luominen on nopeampaa kuin ennen. Tähän työhön tehdyn tutkimuksen perusteella TLS 1.3 on onnistunut siinä, mihin IETF pyrki. Yhteyden luominen saatiin puolet nopeammaksi ja algoritmit, metodit ja funktiot tämän toteuttamiseksi ovat vahvempia kuin koskaan ennen. Työssä tehdyt nopeusmittaukset sekä kättelyprotokollan tutkiminen tukevat tätä väitettä. Neljä vuotta avointa kehitystä isojen ja pienten tahojen toimesta tuottivat protokollan, johon luottavat verkon suurimmat nimet kuten Google, Apple ja Microsoft.

Työn tuloksista käy ilmi erityisesti nopeuteen liittyvät parannukset. Kättelyn kesto saatiin pudotettua kahden edestakaisen matkan ajasta vain yhteen, joka oikeassa maailmassa voi vastata useita sekunteja. Tämä myös vähentää verkkoliikennettä ja prosessointiaikoja jättäen resursseja suuremmalle käyttäjämäärälle sekä mahdollistaen edullisemmat palvelinympäristöt.

Turvallisuutta parantaa uuden protokollan kättelyprosessi, joka sisältää aiempaa enemmän salattuja viestejä. Turvallisuutta parantaa myös tarkkaan valittu lista avaimenvaihtoon ja varmistukseen käytettävästä algoritmeista, salausalgoritmeista sekä viestinvarmistuskoodeista. Aiemmin käytetyt hienolta vaikuttavat algoritmit sekä toteutustavat ovat vaihdettu vankkoihin matemaattisiin algoritmeihin ja varmaksi todettuihin metodeihin. Työssä on ollut mukana niin matemaatikkoja kuin tietotekniikan ja -turvan rautaisia ammattilaisia.

Työn tutkimussuunnitelma ja -metodit olivat aiheen laajuuteen nähden oikein määritelty. Protokollien nopeusvertailu sekä kättelyn tarkastelu riittivät todistamaan, mitä teoriaosuudessa väitettiin. Tarkempi kättelyn tarkastelu esimerkiksi Wireshark-

ohjelmalla olisi ollut teorian toistamista kahteen kertaan, todistaen vain kättelyviestien olevan teorian mukaiset. Ilman tarkempaa matemaattista tutkimista ja todella vahvaa teoreettisen matematiikan taustaa salausalgoritmien sisäisten toimintojen tutkiminen kävi turhan haastavaksi. Jotkin konsepteista ovat helppoja toisten taas vaatiessa satojen tuntien tutkimustyön. Hyvä esimerkki tästä on AES-salauksessa käytettävä "Finite Field Arithmetic", sekä Curve25519-käyrän sisältämät matemaattiset ominaisuudet [118]. Kaiken kaikkiaan työ onnistui kuten suunniteltu, vaikka aikaa kuluikin odotettua enemmän.

Tutkimus osoitti, että uusi TLS-protokolla on valmis yleiseen käyttöön. Nimekkäät yritykset ovat olleet mukana luomistyössä ja tukevat protokollaa omilla tuotteillaan. Näitä tuotteita ovat esimerkiksi Google Chrome, Mozilla Firefox sekä Java. Javan listalla oleminen kielii siitä, että lähes mikä tahansa laite saadaan Internettiin ja käyttämään salattuja yhteyksiä.

Työtä rajoitti tutkimusmetodien rajaaminen laboratorioympäristöön. Esimerkit oikeista ympäristöistä ja nopeuseduista jäivät kokonaan tarkastelematta, sillä protokolla oli vasta julkistettu ja tuki laboratorioympäristön sovelluksiin lisättiin kuukausia virallisen version julkistamisen jälkeen. Oikeat ympäristöt olisivat myös vääristäneet tuloksia riippuen verkkopalvelinten aktiivisuudesta sekä välissä olevasta liikenteestä.

Jatkotutkimuksena tulisi tarkastella protokollan mahdollisia haavoittuvuuksia sekä algoritmien potentiaalisia heikkouksia. Myös kättelyn tarkastelu pakettianalyysaattorilla sekä algoritmien sisäisen toiminnan tarkastelu toisi lisäarvoa tutkimukseen.

LÄHTEET

[1] Eric Rescorla, Datatracker: The Transport Layer Security (TLS) Protocol Version 1.3, Internet Engineering Task Force, 2018, verkkosivu. Saatavilla (viitattu: 30.4.2019): https://datatracker.ietf.org/doc/rfc8446/

[2] Eric Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3, Internet Engineering Task Force, 2018, verkkosivu. Saatavilla (viitattu: 5.5.2019): https://tools.ietf.org/html/rfc8446

[3] Thomas Dierks, Eric Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2, Network Working Group, 2008, verkkosivu. Saatavilla (viitattu: 5.5.2019): https://www.ietf.org/rfc/rfc5246.txt

[4] Google Scholar Results, verkkosivu. Saatavilla (viitattu: 5.5.2019): https://scholar.google.com/scholar?um=1&ie=UTF-8&lr&cites=16225214488298801949

[5] Agathoklis Prodromou, TLS Security 6: Examples of TLS Vulnerabilities and Attacks, Acunetix, 2019, verkkosivu. Saatavilla (viitattu: 5.5.2019): https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/

[6] Tech-FAQ, The OSI Model - What It Is; Why It Matters; Why It Doesn't Matter, verkkosivu. Saatavilla (viitattu: 19.6.2019): http://www.tech-faq.com/osi-model.html

[7] Paul Simoneau, The OSI Model: Understanding the Seven layers of Computer Networks, 2006, verkkosivu. Saatavilla (viitattu: 20.6.2019): http://ru6.cti.gr/bouras-old/WP_Simoneau_OSIModel.pdf

[8] Kirsti Ala-Mutka, Matti Rintala, Vespe Savikko, Jarmo Palviainen, OSI-malli, Tampere University of Technology, 2002, verkkosivu. Saatavilla (viitattu: 8.4.2019): http://www.cs.tut.fi/etaopetus/titepk/luku19/OSI.html

[9] Harju, Väärälä, Puustelli, Kokkonen, OSI-malli, Kouvolan seudun ammattiopisto, 2010, verkkosivu. Saatavilla (viitattu: 8.4.2019): http://www.koudata.fi/node/598

[10] The Linux Information Project, Presentation Layer Definition, 2005, verkkosivu. Saatavilla (viitattu: 8.4.2019): http://www.linfo.org/presentation_layer.html

[11] The Linux Information Project, Application Layer Definition, 2005, verkkosivu. Saatavilla (viitattu: 8.4.2019): http://www.linfo.org/application_layer.html

[12] Kevin R. Fall, W. Richard Stevens, TCP/IP Illustrated, Volume 1, Addison-Wesley Professional, 2011, sivut 15-17, 579-581. Saatavilla (viitattu: 10.4.2019): http://www.r-5.org/files/books/computers/internals/net/Richard_Stevens-TCP-IP_Illustrated-EN.pdf

[13] Margaret Rouse, TCP (Transmission Control Protocol), verkkosivu. Saatavilla (viitattu: 10.4.2019): https://searchnetworking.techtarget.com/definition/TCP

[14] Douglas E. Comer, INTERNETWORKING with TCP/IP: Principles, Protocols, and Architectures, Prentice Hall, 1988, sivut 217-220. Saatavilla (viitattu: 3.4.2019): https://doc.lagout.org/network/Internetworking%20with%20TCP_IP%20%20Vol%20I.pdf

[15] Charles M. Kozierok, The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference, No Starch Press, 2005, sivu 881. Saatavilla (viitattu: 4.4.2019): https://lira.epac.to/DOCS-TECH/Networking/The%20TCP-IP%20Guide.pdf

[16] Charles M. Kozierok, TCP Connection Preparation: Transmission Control Blocks (TCBs) and Passive and Active Socket OPENs , 2005, verkkosivu. Saatavilla (viitattu: 4.4.2019): http://tcpipguide.com/free/t_TCPConnectionPreparationTransmissionControlBlocksT.htm

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol -- HTTP/1.1, RFC 2616, DOI 10.17487/RFC2616, 1999, sivu 6. Saatavilla (viitattu: 4.4.2019): https://www.rfc-editor.org/info/rfc2616

[18] David, Difference between HTTP 1.0 and 1.1, Differencebetween, 2013, verkkosivu. Saatavilla (viitattu: 6.4.2019): http://www.differencebetween.net/technology/protocols-formats/difference-between-http-1-0-and-1-1/

[19] J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol -- HTTP/1.1: Changes from HTTP/1.0, Greenbytes, 1999, verkkosivu. Saatavilla (viitattu: 30.12.2018): https://greenbytes.de/tech/webdav/rfc2616.html#rfc.section.19.6.1

[20] J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol -- HTTP/1.1, Network Working Group, 1999, verkkosivu. Saatavilla (viitattu: 30.12.2018): https://www.ietf.org/rfc/rfc2616.txt

[21] Gokulakrishnan Kalaikovan, Comparison of HTTP and HTTP/2, 2017, verkkosivu. Saatavilla (viitattu: 1.1.2019): https://gokul.site/blog/2017/http-vs-http2/

[22] Peter Bright, HTTP/2 finished, coming to browsers within weeks, ArsTechnica, 2015, verkkosivu. Saatavilla (viitattu: 7.4.2019): https://arstechnica.com/information-technology/2015/02/http2-finished-coming-to-browsers-within-weeks/

[23] Internet Engineering Task Force, Hypertext Transfer Protocol Version 2 (HTTP/2), Internet Engineering Task Force, 2014, verkkosivu. Saatavilla (viitattu: 7.4.2019): https://datatracker.ietf.org/doc/draft-ietf-httpbis-http2/history/#history-404477

[24] Mark Nottingham, Barry Leiba, HTTP/2 Approved, Internet Engineering Task Force, 2015, verkkosivu. Saatavilla (viitattu: 7.4.2019): https://www.ietf.org/blog/http2-approved/

[25] M. Belshe, R. Peon, M. Thomson, Hypertext Transfer Protocol Version 2 (HTTP/2), Internet Engineering Task Force, 2015, verkkosivu. Saatavilla (viitattu: 7.4.2019): https://tools.ietf.org/html/rfc7540

[26] Jerome Louvel, Can the rise of SPDY threaten HTTP?, Restlet, 2011, verkkosivu. Saatavilla (viitattu: 7.4.2019): http://blog.restlet.com/2011/10/06/can-the-rise-of-spdy-threaten-http/

[27] Alexis Deveria, Can I use http2?, verkkosivu. Saatavilla (viitattu: 7.4.2019): https://caniuse.com/#search=http2

[28] Usage statistics of HTTP/2 for websites, W3Techs, 2019, verkkosivu. Saatavilla (viitattu: 7.4.2019): https://w3techs.com/technologies/details/ce-http2/all/all

[29] Ilya Grigorik, High Performance Browser Networking: HTTP/2, O'Reilly Media, 2013, verkkosivu. Saatavilla (viitattu: 7.4.2019): https://hpbn.co/http2/

[30] Michael Pratt, HTTP/2.0 Initial Draft Released, ApiUx, 2013, verkkosivu. Saatavilla (viitattu: 7.4.2019): http://apiux.com/2013/07/23/http2-0-initial-draft-released/

[31] Dio Synodinos, HTTP 2.0 First Draft Published, InfoQ, 2012, verkkosivu. Saatavilla (viitattu: 8.4.2019): https://www.infoq.com/news/2012/11/http20-first-draft/

[32] Javier Garza, How does HTTP/2 solve the Head of Line blocking (HOL) issue, Akamai, 2017, verkkosivu. Saatavilla (viitattu: 8.4.2019): https://community.akamai.com/customers/s/article/How-does-HTTP-2-solve-the-Head-of-Line-blocking-HOL-issue

[33] Zeus Kerravala, What is Transport Layer Security (TLS)?, Network World, 2018, verkkosivu. Saatavilla (viitattu: 13.5.2019): https://www.networkworld.com/article/2303073/lan-wan-what-is-transport-layer-security-protocol.html

[34] Rohit, Elliptic Curve Cryptography: A Case for Mobile Encryption, SecurityLearn, 2014, verkkosivu. Saatavilla (viitattu: 11.5.2019): http://www.securitylearn.net/tag/mobile-encryption/

[35] Internet Society, TLS Basics, Internet Society , verkkosivu. Saatavilla (viitattu: 20.5.2019): https://www.internetsociety.org/deploy360/tls/basics/

[36] John Carl Villanueva, An Introduction To Cipher Suites, JScape, 2018, verkkosivu. Saatavilla (viitattu: 8.6.2019): https://www.jscape.com/blog/cipher-suites

[37] Avinash Kak, Lecture 8: AES: The Advanced Encryption Standard, Purdue University, 2019, verkkosivu. Saatavilla (viitattu: 8.6.2019): https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf

[38] The Information Warfare Site, Descriptions of SHA-256, SHA-384, and SHA-512, 2001, verkkosivu. Saatavilla (viitattu: 8.6.2019): http://www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf

[39] Mihir Bellare, Chanathip Namprempre, Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm, 2007, verkkosivu. Saatavilla (viitattu: 8.6.2019): https://cseweb.ucsd.edu/~mihir/papers/oem.pdf

[40] Andy Brodie, Overview of TLS v1.3, OWASP, 2018, .verkkosivu, 19-20. Saatavilla (viitattu: 8.6.2019): https://www.owasp.org/images/9/91/OWASPLondon20180125_TLSv1.3_Andy_Brodie.pdf

[41] Michael Driscoll, The New Illustrated TLS Connection, 2018, verkkosivu. Saatavilla (viitattu: 8.6.2019): https://tls13.ulfheim.net/

[42] Chris Kowalczyk, Message Authentication Code (MAC), Crypto-IT, 2013, verkkosivu. Saatavilla (viitattu: 10.6.2019): http://www.crypto-it.net/eng/theory/mac.html

[43] IBM Knowledge Center, Digital signatures in SSL and TLS, IBM, 2019, verkkosivu. Saatavilla (viitattu: 10.6.2019): https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10520_.htm

[44] Dobromir Todorov, Mechanics of User Identification and Authentication, CRC Press, 2007, sivu 452. Saatavilla (viitattu: 8.6.2019): https://books.google.fi/books?redir_esc=y&hl=fi&id=eIPA4v0u05EC&q=Master+key#v=snippet&q=Master%20key&f=false

[45] Rolf Oppliger, SSL and TLS: Theory and Practice, Second Edition, Artech House, 2016, sivu 97. Saatavilla (viitattu: 8.6.2019): https://books.google.fi/books?id=jm6uDgAAQBAJ

[46] Manuel Mogollon, Cryptography and Security Services: Mechanisms and Applications: Mechanisms and Applications, IGI Global, 2008, sivu 85-86. Saatavilla (viitattu: 8.6.2019): https://books.google.fi/books?hl=fi&id=jgJQce_GRyEC

[47] Michael Driscoll, The Illustrated TLS Connection, 2018, verkkosivu. Saatavilla (viitattu: 12.5.2019): https://tls.ulfheim.net/

[48] Kenny Paterson, Authenticated key Exchange (in TLS), Royal Holloway University of Londong, 2015, verkkosivu. Saatavilla (viitattu: 12.5.2019): https://summerschool-croatia.cs.ru.nl/2015/Authenticated%20key%20exchange.pdf

[49] David Mazières, SSL/TLS Overview, Stanford University , verkkosivu. Saatavilla (viitattu: 12.5.2019): http://www.scs.stanford.edu/nyu/04sp/notes/l22.pdf

[50] Jim Willeke, Master Secret, LDAPWiki, 2017, verkkosivu. Saatavilla (viitattu: 12.5.2019): https://ldapwiki.com/wiki/Master%20Secret

[51] Gabrielle Hempel, TLS 1.2 and TLS 1.3 - What's the Difference?, Cybrary, 2018, verkkosivu. Saatavilla (viitattu: 20.5.2019): https://www.cybrary.it/0p3n/tls-versions-the-difference/

[52] John Kennedy, Michael Satran, TLS Record Protocol, Microsoft, 2018, verkkosivu. Saatavilla (viitattu: 20.5.2019): https://docs.microsoft.com/en-us/windows/desktop/secauthn/tls-record-protocol

[53] Richard Levitte, Rich Salz, SSL_alert_type_string, OpenSSL, 2018, verkkosivu. Saatavilla (viitattu: 20.5.2019): https://www.openssl.org/docs/man1.1.0/man3/SSL_alert_type_string.html

[54] GnuTLS, The TLS alert protocol, GnuTLS Project , verkkosivu. Saatavilla (viitattu: 20.5.2019): https://www.gnutls.org/manual/html_node/The-TLS-Alert-Protocol.html

[55] Chris Conlon, Eric Blankenhorn, Kaleb Himes, Michael Pollard, A Comparison of Differences in TLS 1.1 and TLS 1.2, WolfSSL, 2015, verkkosivu. Saatavilla (viitattu: 20.5.2019): https://www.wolfssl.com/a-comparison-of-differences-in-tls-1-1-and-tls-1-2/

[56] Tim Dierks, C. Allen, The TLS Protocol Version 1.0, Network Working Group, 1999, verkkosivu. Saatavilla (viitattu: 10.1.2019): https://tools.ietf.org/html/rfc2246

[57] Tim Dierks, Security Standards and Name Changes in the Browser Wars, 2014, verkkosivu. Saatavilla (viitattu: 13.5.2019): http://tim.dierks.org/2014/05/security-standards-and-name-changes-in.html

[58] Tim Dierks, Eric Rescorla, The Transport Layer Security (TLS) Protocol Version 1.1, Network Working Group, 2006, verkkosivu. Saatavilla (viitattu: 13.5.2019): https://www.ietf.org/rfc/rfc4346.txt

[59] Joshua Davies, An Illustrated Guide to the BEAST Attack, Command Line Fanatic, 2014, verkkosivu. Saatavilla (viitattu: 19.2.2019): https://commandlinefanatic.com/cgi-bin/showarticle.cgi?article=art027

[60] Taylor Hornby, Crypto Noobs #1: Initialization Vectors, Crypto Fails, 2013, verkkosivu. Saatavilla (viitattu: 19.2.2019): https://www.cryptofails.com/post/70059609995/crypto-noobs-1-initialization-vectors

[61] What is an initialization vector?, TechTarget, 2011, verkkosivu. Saatavilla (viitattu: 19.2.2019): https://whatis.techtarget.com/definition/initialization-vector-IV

[62] Filippo Valsorda, TLS nonce-nse, CloudFlare, 2016, verkkosivu. Saatavilla (viitattu: 19.2.2019): https://blog.cloudflare.com/tls-nonce-nse/

[63] Tim Polk, kerry McKay, Santosh Chokhani, Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations, NIST, 2013, NIST Special Publication 800-52. Saatavilla (viitattu: 19.2.2019): http://dx.doi.org/10.6028/NIST.SP.800-52r1

[64] Thomas Pornin, OpenSSL TLS 1.1 Cipher Suites, StackExchange, 2014, verkkosivu. Saatavilla (viitattu: 9.6.2019): https://security.stackexchange.com/questions/65130/openssl-tls-1-1-cipher-suites

[65] OpenSSL Authors, Ciphers, OpenSSL Project, 2018, verkkosivu. Saatavilla (viitattu: 9.6.2019): https://www.openssl.org/docs/manmaster/man1/ciphers.html

[66] Chris Conlon, Eric Blankenhorn, Kaleb Himes, Michael Pollard, A Comparison of Differences in TLS 1.1, TLS 1.2 and TLS 1.3, WolfSSL, 2018, verkkosivu. Saatavilla (viitattu: 20.5.2019): https://www.wolfssl.com/a-comparison-of-differences-in-tls-1-1-and-tls-1-2-2/

[67] The Heartbleed Bug, Synopsys, 2014, verkkosivu. Saatavilla (viitattu: 19.2.2019): http://heartbleed.com/

[68] Adam Langley, PKCS#1 Signature Validation, 2014, verkkosivu. Saatavilla (viitattu: 19.2.2019): https://www.imperialviolet.org/2014/09/26/pkcs1.html

[69] Paul Ducklin, Anamoty of a "goto fail", Sophos, 2014, verkkosivu. Saatavilla (viitattu: 19.2.2019): https://nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/

[70] Adam Langley, The POODLE bites again, 2014, verkkosivu. Saatavilla (viitattu: 19.2.2019): https://www.imperialviolet.org/2014/12/08/poodleagain.html

[71] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, Paul Zimmermann, Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice, 2015, verkkosivu. Saatavilla (viitattu: 19.2.2019): https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf

[72] Jason Sabin, Logjam Attack: What You Need to Know, Digicert, 2015, verkkosivu. Saatavilla (viitattu: 19.2.2019): https://www.digicert.com/blog/logjam-attack/

[73] Nick Sullivan, A Detailed Look at RFC 8446 (a.k.a. TLS 1.3), CloudFlare, 2018, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/

[74] Eric Rescorla, TLS 1.3 Wish List, Internet Engineering Task Force, 2013, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://www.ietf.org/proceedings/87/slides/slides-87-tls-5.pdf

[75] Margaret Rouse, cut-and-paste attack, TechTarget, 2007, verkkosivu. Saatavilla (viitattu: 19.2.2019): https://searchsecurity.techtarget.com/definition/cut-and-paste-attack

[76] Eric Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3 draft-ietf-tls-tls13-2: 0-RTT and Anti-Replay, Internet Engineering Task Force, 2017, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://tools.ietf.org/html/draft-ietf-tls-tls13-21#section-8

[77] Mihir Bellare, Phillip Rogaway, PSS: Provably Secure Encoding Method for Digital Signatures, IEEE, 1998, verkkosivu. Saatavilla (viitattu: 1.6.2019): https://web.archive.org/web/20170810025803/http://grouper.ieee.org/groups/1363/P1363a/contributions/pss-submission.pdf

[78] Chris Conlon, Eric Blankenhorn, Kaleb Himes, Michael Pollard, Differences between TLS 1.2 and TLS 1.3, WolfSSL, 2017, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://www.wolfssl.com/differences-between-tls-1-2-and-tls-1-3/

[79] Jesse Victors, TLS 1.3 and the future of cryptographic protocols, Synopsys, 2016, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://www.synopsys.com/blogs/software-security/tls-1-3/

[80] Daniel J. Bernstein, Boring crypto, 2015, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://cr.yp.to/talks/2015.10.05/slides-djb-20151005-a4.pdf

[81] A. Langley, W. Chang, N. Mavrogiannopoulos, J. Strombergson, S. Josefsson, ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS) draft-ietf-tls-chacha20-poly1305-04, Internet Engineering Task Force, 2015, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://tools.ietf.org/html/draft-ietf-tls-chacha20-poly1305-04

[82] Daniel J. Bernstein, Index of formal scientific papers, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://cr.yp.to/papers.html

[83] Daniel J. Bernstein, Curve25519: new Diffie-Hellman speed records, 2006, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://cr.yp.to/ecdh/curve25519-20060209.pdf

[84] Daniel J. Bernstein, ChaCha, a variant of Salsa20, The University of Illinois at Chicago, 2008, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://cr.yp.to/chacha/chacha-20080128.pdf

[85] Daniel J. Bernstein, The Poly1305-AES message-authentication code, The University of Illinois at Chicago, 2005, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://cr.yp.to/mac/poly1305-20050329.pdf

[86] Ilya Grigorik, High Performance Browser Networking: Transport Layer Security (TLS), O'Reilly Media, 2013, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://www.oreilly.com/library/view/high-performance-browser/9781449344757/ch04.html

[87] Agathoklis Prodromou, TLS Security 5: Establishing a TLS Connection, Acunetix, 2019, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://www.acunetix.com/blog/articles/establishing-tls-ssl-connection-part-5/

[88] Jay Thakkar, TLS 1.3 Handshake: Taking a Closer Look, The SSL Store, 2018, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://www.thesslstore.com/blog/tls-1-3-handshake-tls-1-2/

[89] Pratik Guha Sarkar, Shawn Fitzgerald, Attacks on SSL: A Comprehensive Study of BEAST, CRIME, TIME, BREACH, LUCKY13 & RC4 biases, NCC Group, 2013, verkkosivu. Saatavilla (viitattu: 19.2.2019): https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/ssl_attacks_survey.pdf

[90] Dennis Fisher, CRIME Attack Uses Compression Ratio of TLS Requests as Side Channel to Hijack Secure Sessions, Threatpost, 2012, verkkosivu. Saatavilla (viitattu: 19.2.2019): https://threatpost.com/crime-attack-uses-compression-ratio-tls-requests-side-channel-hijack-secure-sessions-091312/77006/

[91] Greg Linden, Marissa Mayer at Web 2.0, 2006, verkkosivu. Saatavilla (viitattu: 19.4.2019): http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html

[92] James Hamilton, The Cost of latency, 2009, verkkosivu. Saatavilla (viitattu: 19.4.2019): https://perspectives.mvdirona.com/2009/10/the-cost-of-latency/

[93] Billy Hoffman, SSL Performance Diary #4: Optimizing the TLS Handshake, Zoompf, 2014, verkkosivu. Saatavilla (viitattu: 1.2.2019): https://zoompf.com/blog/2014/12/optimizing-tls-handshake/

[94] A. Langley, N. Modadugu, B. Moeller, Transport Layer Security (TLS) False Start, Internet Engineering Task Force, 2016, verkkosivu. Saatavilla (viitattu: 1.2.2019): https://tools.ietf.org/html/rfc7918

[95] Jim Willeke, 0-RTT Handshakes, LDAPWiki, 2018, verkkosivu. Saatavilla (viitattu: 19.4.2019): https://ldapwiki.com/wiki/0-RTT%20Handshakes

[96] Ilya Grigori, Networking 101: Chapter 14 - Transport Layer Security (TLS), O'Reilly Media, 2013, verkkosivu. Saatavilla (viitattu: 31.5.2019): https://hpbn.co/transport-layer-security-tls/

[97] Bruce Schneier, NIST Defines New Versions of SHA-512, Schneier on Security, 2011, verkkosivu. Saatavilla (viitattu: 16.7.2019): https://www.schneier.com/blog/archives/2011/02/nist_defines_ne.html

[98] Diego Assencio, Performance of cryptographic algorithms in OpenSSL, 2014, verkkosivu. Saatavilla (viitattu: 4.6.2019): https://diego.assencio.com/?index=0e18f5485ff4de58d94e494e3649c6eb

[99] A. Langley, M. Hamburg, S. Turner, sn3rd, Elliptic Curves for Security, Internet Engineering Task Force, 2016, verkkosivu. Saatavilla (viitattu: 4.6.2019): https://tools.ietf.org/html/rfc7748

[100] Gregory Maxwell, NIST approved crypto in Tor?, TOR Project, 2013, verkkosivu. Saatavilla (viitattu: 22.2.2019): https://lists.torproject.org/pipermail/tor-talk/2013-September/029956.html

[101] John Kelsey, Dual EC in X9.82 and SP 800-90, NIST, 2014, verkkosivu. Saatavilla (viitattu: 22.2.2019): https://csrc.nist.gov/csrc/media/projects/crypto-standards-development-process/documents/dualec_in_x982_and_sp800-90.pdf

[102] Transition Plans for Key Establishment Schemes using Public Key Cryptography , NIST, 2017, verkkosivu. Saatavilla (viitattu: 22.2.2019): https://csrc.nist.gov/News/2017/Transition-Plans-for-Key-Establishment-Schemes

[103] Michael Düll, Björn Haase, Gesine Hinterwälder, Michael Hutter, Christof Paar, Ana Helena Sánchez, Peter Schwabe, High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers: Designs, Codes and Cryptography, Volume 77, Springer, 2015, sivut 493-514. Saatavilla (viitattu: 22.2.2019): https://link.springer.com/article/10.1007/s10623-015-0087-1

[104] S. Josefsson, M. Pegourie-Gonnard, Curve25519 and Curve448 for Transport Layer Security (TLS) draft-ietf-tls-curve25519-01, Internet Engineering Task Force, 2015, verkkosivu. Saatavilla (viitattu: 25.4.2019): https://tools.ietf.org/html/draft-ietf-tls-curve25519-01

[105] Moxie Marlinspike, Open Whisper Systems partners with Google on end-to-end encryption for Allo, Signal, 2016, verkkosivu. Saatavilla (viitattu: 25.4.2019): https://signal.org/blog/allo/

[106] Moxie Marlinspike, WhatsApp's Signal Protocol integration is now complete, Signal, 2016, verkkosivu. Saatavilla (viitattu: 25.4.2019): https://signal.org/blog/whatsapp-complete/

[107] WebPageTest, 2018, verkkosivu. Saatavilla (viitattu: 10.10.2018): https://www.webpagetest.org/

[108] SSLSessionCacheTimeout Directive, Apache HTTP Server Project, 2018, verkkosivu. Saatavilla (viitattu: 10.10.2018): https://httpd.apache.org/docs/2.4/mod/mod_ssl.html#sslsessioncachetimeout

[109] Google Toolbox: HAR Analyzer, Google, 2018, verkkosivu. Saatavilla (viitattu: 10.10.2018): https://toolbox.googleapps.com/apps/har_analyzer/

[110] Jonathan Lee, Tobias Lidskog, Peter Hedenskog, Compare Sitespeed, 2018, verkkosivu. Saatavilla (viitattu: 10.10.2018): https://compare.sitespeed.io/

[111] Jan Odvarko, HTTP Archive Viewer 2.0.17, verkkosivu. Saatavilla (viitattu: 10.10.2018): http://www.softwareishard.com/har/viewer/

[112] Chrome Dev Tools, Google, 2018, verkkosivu. Saatavilla (viitattu: 10.10.2018): https://developers.google.com/web/tools/chrome-devtools/

[113] HTTP/1.1 vs HTTP/2: What's the Difference?, Digital Ocean, 2019, verkkosivu. Saatavilla (viitattu: 7.6.2019): https://www.digitalocean.com/community/tutorials/http-1-1-vs-http-2-what-s-the-difference

[114] Equifax had 'admin' as login and password in Argentina, BBC News, 2017, verkkosivu. Saatavilla (viitattu: 7.6.2019): https://www.bbc.com/news/technology-41257576

[115] Alfred NG, How the Equifax hack happened, and what still needs to be done, Cnet, 2018, verkkosivu. Saatavilla (viitattu: 7.6.2019): https://www.cnet.com/news/equifaxs-hack-one-year-later-a-look-back-at-how-it-happened-and-whats-changed/

[116] Actions Taken by Equifax and Federal Agencies in Response to the 2017 Breach, U.S. Government Accountability Office, 2018, verkkosivu. Saatavilla (viitattu: 7.6.2019): https://www.gao.gov/assets/700/694158.pdf

[117] David Shepardson, Equifax failed to patch security vulnerability in March: former CEO, Reuters, 2017, verkkosivu. Saatavilla (viitattu: 7.6.2019): https://www.reuters.com/article/us-equifax-breach/equifax-failed-to-patch-security-vulnerability-in-march-former-ceo-idUSKCN1C71VY

[118] Christoforus Juan Benvenuto, Galous Field in Cryptography, University of Washington, 2012, verkkosivu. Saatavilla (viitattu: 7.6.2019): https://sites.math.washington.edu/~morrow/336_12/papers/juan.pdf

LIITTEET

Liite A: Openssl speed rsa2048 / Openssl speed rsa4096

```
OpenSSL 1.1.1b  26 Feb 2019
built on: Thu Feb 28 09:24:32 2019 UTC

mox@lakka:~$
openssl speed rsa2048

Doing 2048 bits private rsa's for 10s: 3205 2048 bits private
RSA's in 9.91s
Doing 2048 bits public rsa's for 10s: 113938 2048 bits public
RSA's in 10.00s

                  sign    verify    sign/s verify/s
rsa 2048 bits 0.003092s 0.000088s    323.4  11393.8

mox@lakka:~$
openssl speed rsa4096

Doing 4096 bits private rsa's for 10s: 494 4096 bits private
RSA's in 9.74s
Doing 4096 bits public rsa's for 10s: 34095 4096 bits public
RSA's in 9.99s

                  sign    verify    sign/s verify/s
rsa 4096 bits 0.019717s 0.000293s     50.7   3412.9
```

Liite B: Openssl speed aes-128-cbc / Openssl speed aes-256-cbc

```
OpenSSL 1.1.1b  26 Feb 2019
built on: Thu Feb 28 09:24:32 2019 UTC
```

mox@lakka:~$
openssl speed aes-128-cbc

```
Doing aes-128 cbc for 3s on 16 size blocks: 13790543 aes-128 cbc's in 2.91s
Doing aes-128 cbc for 3s on 64 size blocks: 4164619 aes-128 cbc's in 2.93s
Doing aes-128 cbc for 3s on 256 size blocks: 1089278 aes-128 cbc's in 2.98s
Doing aes-128 cbc for 3s on 1024 size blocks: 691105 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 8192 size blocks: 86694 aes-128 cbc's in 2.99s

The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes     256 bytes   1024 bytes   8192 bytes
aes-128 cbc      75824.29k    90967.79k    93575.56k   235897.17k   237524.16k
```

mox@lakka:~$
openssl speed aes-256-cbc

```
Doing aes-256 cbc for 3s on 16 size blocks: 11211800 aes-256 cbc's in 2.99s
Doing aes-256 cbc for 3s on 64 size blocks: 2982703 aes-256 cbc's in 2.99s
Doing aes-256 cbc for 3s on 256 size blocks: 776805 aes-256 cbc's in 2.99s
Doing aes-256 cbc for 3s on 1024 size blocks: 497881 aes-256 cbc's in 2.97s
Doing aes-256 cbc for 3s on 8192 size blocks: 61862 aes-256 cbc's in 2.93s

The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes     256 bytes   1024 bytes   8192 bytes
aes-256 cbc      59996.25k    63843.81k    66509.06k   171659.98k   172960.24k
```